



# INSIDE MACINTOSH

---

## QuickDraw GX Printing



**Addison-Wesley Publishing Company**

Reading, Massachusetts   Menlo Park, California   New York  
Don Mills, Ontario   Wokingham, England   Amsterdam   Bonn  
Sydney   Singapore   Tokyo   Madrid   San Juan  
Paris   Seoul   Milan   Mexico City   Taipei

Apple Computer, Inc.  
© 1994 Apple Computer, Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple Macintosh computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for printing or clerical errors.

Apple Computer, Inc.  
20525 Mariani Avenue  
Cupertino, CA 95014  
408-996-1010

APDA, Apple, the Apple logo, LaserWriter, Macintosh, and StyleWriter are trademarks of Apple Computer, Inc., registered in the United States and other countries.

ColorSync, Finder, and QuickDraw are trademarks of Apple Computer, Inc.

Adobe Illustrator, Adobe Photoshop, and PostScript are trademarks of Adobe Systems Incorporated, which may be registered in certain jurisdictions.

America Online is a service mark of Quantum Computer Services, Inc.

CompuServe is a registered service mark of CompuServe, Inc.

FrameMaker is a registered trademark of Frame Technology Corporation.

Helvetica and Palatino are registered trademarks of Linotype Company.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Optrotech is a trademark of Orbotech Corporation.

Simultaneously published in the United States and Canada.

#### LIMITED WARRANTY ON MEDIA AND REPLACEMENT

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

ISBN 0-201-40677-2  
1 2 3 4 5 6 7 8 9-CRW-9897969594  
First Printing, June 1994

The paper used in this book meets the EPA standards for recycled fiber.

#### Library of Congress Cataloging-in-Publication Data

Inside Macintosh. QuickDraw GX printing / [by Apple Computer, Inc.].  
p. cm.  
Includes index.  
ISBN 0-201-40677-2  
1. Macintosh (Computer)—Programming. 2. Computer graphics.  
3. QuickDraw GX. I. Apple Computer, Inc. II. Title: QuickDraw GX printing.  
QA76.8.M31562282 1994  
005.7'1265—dc20

94-17336  
CIP

# Contents

Figures, Tables, and Listings    xi

Preface

About This Book    xv

---

What to Read    xvi  
Chapter Organization    xvii  
Conventions Used in This Book    xviii  
    Special Fonts    xviii  
    Types of Notes    xviii  
    Numerical Formats    xviii  
    Type Definitions for Enumerations    xix  
    Illustrations    xix  
Development Environment    xix  
For More Information    xx

Chapter 1

Introduction to Printing With QuickDraw GX    1-1

---

About QuickDraw GX Printing    1-3  
    Core Printing-Related Objects    1-6  
    Desktop Printers    1-7  
    Print Files    1-8  
    Printer Drivers    1-8  
    Printing Extensions    1-9  
    Dialog Boxes    1-10  
    Message Passing    1-13  
About QuickDraw GX Printing-Related Objects    1-16  
    Job Objects    1-16  
    Format Objects    1-17  
    Paper-Type Objects    1-18  
    Collection Objects    1-18  
    Printer Objects    1-20  
    Print File Objects    1-20  
    Summary of QuickDraw GX Printing-Related Objects    1-20  
Using Printing-Related Objects With Other QuickDraw GX Objects    1-23  
    Shape Objects    1-23  
    Tag Objects    1-24  
    View Port Objects    1-24  
    View Device Objects    1-25

Implementing QuickDraw GX Printing Features	1-25
Core Printing Features	1-26
Customizing QuickDraw GX Printing Features	1-28
Advanced Printing Features	1-30
Compatibility With the Macintosh Printing Manager	1-30

## Chapter 2

## Core Printing Features 2-1

---

About Core Printing Features	2-3
Core Print Objects	2-5
Job Object Properties	2-5
Format Object Properties	2-7
Paper-Type Object Properties	2-8
Edit Menu Structure	2-9
Using Core Printing Features	2-10
Initializing QuickDraw GX Printing	2-11
Creating a Job Object for a Printable Document	2-12
Error Handling	2-14
Supporting QuickDraw GX Print Dialog Boxes	2-17
Printing Documents Using QuickDraw GX	2-20
Printing Pages as Single Picture Shapes	2-21
Printing Pages by Capturing Shapes	2-22
Saving a Job Object With a Document File	2-24
Saving a Job Object in a Single Handle	2-25
Saving a Job Object Using a Flattening Function	2-27
Disposing of a Job Object When Closing a Document	2-28
Retrieving a Job Object When Opening a Document	2-29
Retrieving a Job Object From a Handle	2-30
Retrieving a Job Object Using an Unflattening Function	2-32
Obtaining Object References	2-33
Obtaining Information From a Format Object	2-33
Displaying QuickDraw GX Print Dialog Boxes	2-35
Displaying the Page Setup Dialog Box	2-35
Displaying the Print Dialog Box	2-37
Supporting Printing From the Finder	2-39
Updating Job Object Information	2-42
Printing Macintosh Printing Manager Documents	2-44
Core Printing Features Reference	2-46
Constants and Data Types	2-46
Gestalt Selectors for Printing	2-47
QuickDraw GX Printing-Related Objects	2-47
Edit Menu Location	2-48
Dialog Box Results	2-48

Functions	2-49	
Initializing and Terminating QuickDraw GX Printing Features		2-50
Handling Errors	2-52	
Creating and Managing Job Objects	2-54	
Printing With QuickDraw GX	2-61	
Obtaining Information on Printing-Related Objects		2-68
Displaying the Page Setup and Print Dialog Boxes		2-71
Converting a Print Record	2-75	
Application-Defined Functions	2-76	
Message Override Functions	2-76	
Flattening and Unflattening Functions for Job Objects		2-77
Summary of Core Printing Features	2-79	

---

Chapter 3	Page Formatting and Dialog Box Customization	3-1
-----------	--	-----

---

About Page Formatting and Dialog Box Customization	3-6
About Collection Objects	3-7
Collection Tag IDs and Item IDs	3-7
Item Structures	3-8
Categories of Collection Items	3-9
The Job Collection	3-10
The Format Collection	3-12
The Paper-Type Collection	3-14
About Page Formatting	3-15
Manipulating Format Objects	3-16
Mapping for Format Objects	3-18
Forms and Format Objects	3-20
Halftones and Format Collections	3-21
Dialog Box Customization	3-22
The Dialog Box Panel Resource	3-24
Responding to Panel Events	3-25
Automating Panel Events	3-25
Using Printing-Related Collection Objects	3-27
Accessing Data From a Collection Object	3-28
Using a Collection to Implement the Print One Copy Menu Item	3-29
Replacing Items in Collections	3-31
Specifying Page Ranges in the Job Collection	3-33
Using Format Objects and Collection Items to Format Pages	3-39
Creating a Format Object for a Page in a Document	3-40
Sharing Formats for Document Pages	3-44
Disposing of a Format Object for a Page in a Document	3-47
Using Forms With Format Objects	3-50
Storing Halftone Information in a Format Collection	3-52
Copying a Format Object for Use in Other Documents	3-54
Obtaining the Mapping From a Format Object	3-57
Obtaining a Paper-Type Object Associated With a Format	3-57

Scanning Through a Job's Format Objects	3-59
Associating Format Objects With Document Pages	3-61
Customizing QuickDraw GX Dialog Boxes	3-66
Adding Panels to Dialog Boxes	3-67
Setting Up Dialog Box Resources	3-70
Parsing Page Ranges	3-73
Page Formatting and Dialog Box Customization Reference	3-75
Constants for Loop Status Information	3-76
Constants for Collection Item Categories and Tag IDs	3-76
Collection Item Categories	3-76
Collection Tag ID	3-77
Constants and Data Types for Job Collection Items	3-78
Print-Job Information	3-78
Collation Information	3-80
Copies Information	3-81
Page-Range Information	3-81
Quality Information	3-83
File-Destination Information	3-83
File-Location Information	3-84
File-Format Information	3-84
File-Fonts Information	3-85
Paper-Feed Information	3-85
Manual-Feed Information	3-86
Standard Mapping Information	3-86
Special Mapping Information	3-87
Tray-Mapping Information	3-88
Print-Panel Information	3-88
Format-Panel Information	3-88
Paper-Mapping Information	3-89
Translated-Document Information	3-89
Constants and Data Types for Format Collection Items	3-89
Orientation Information	3-89
Scaling Information	3-91
Direct-Mode Information	3-91
Format-Halftone Information	3-92
Page-Inversion Information	3-92
Horizontal Page-Flip Information	3-93
Vertical Page-Flip Information	3-93
Precise-Bitmap Information	3-93
Paper-Type Lock Information	3-94
Constants and Data Types for Paper-Type Collection Items	3-94
Base Information	3-94
Creator Information	3-95
Units Information	3-96
Flags Information	3-97
Comment Information	3-97

Panel-Related Constants and Data Types	3-98
The Panel Information Structure	3-98
Panel Events	3-99
Panel Responses	3-100
Panel Event Actions	3-101
The Panel Setup Structure	3-101
Printing Panel Kinds	3-102
Parse Range Results	3-102
Functions	3-103
Creating and Manipulating Format Objects	3-103
Manipulating Format Object Properties	3-109
Displaying the Custom Page Setup Dialog Box	3-113
Working With Panels	3-114
Accessing Printing-Related Collection Objects	3-117
Application-Defined Functions	3-119
Message Override Functions for Customizing QuickDraw GX Dialog Boxes	3-119
Looping Through Format Objects	3-126
Dialog Box-Related Resources	3-127
The Panel Resource	3-127
The Extended Item List Resource	3-128
Summary of Page Formatting and Dialog Box Customization	3-133

---

Chapter 4	Advanced Printing Features	4-1
-----------	----------------------------	-----

---

About Advanced Printing Features	4-5
Printer Objects	4-6
Printer Driver Types	4-7
Printer View Devices	4-8
Color Matching for Printers	4-9
Print File Objects	4-9
Synonyms	4-11
General-Purpose PostScript Operator Synonym	4-12
PostScript Control Information Synonym	4-13
Dash Synonym	4-14
Line Cap Synonym	4-14
Halftone Synonym	4-15
Pattern Synonym	4-17
Cubic Synonym	4-17
QuickDraw Picture Synonym	4-18
Printing Modes	4-19
Pen Tables for Vector Devices	4-20

Using Advanced Printing Features	4-21
Using Advanced Job Object Functions	4-21
Obtaining Printer and Printer Driver Information for a Job	4-22
Getting and Setting the Reference Constant for a Job Object	4-23
Copying Job Object Information	4-25
Working With Printer Objects	4-25
Determining a Printer's Resolution	4-26
Retrieving the Color Profile and Color Space for a Printer	4-27
Manipulating Print File Objects	4-29
Opening and Closing a Print File	4-29
Saving a Print File	4-30
Obtaining the Job Object for a Print File	4-30
Reading Print File Data	4-30
Counting the Pages in a Print File	4-31
Adding or Deleting Print File Pages	4-31
Defining Different Paper Sizes	4-31
Creating a Paper-Type Object	4-32
Obtaining the Name of a Paper Type	4-32
Obtaining the Dimensions of a Paper Type	4-33
Scanning the Paper Types Available to a Job	4-34
Implementing Direct-Mode Printing	4-35
Formatting for Text Job Format Mode Printing	4-36
Using Synonyms	4-38
Advanced Printing Features Reference	4-38
Constants and Data Types for Advanced Printing Features	4-39
Job Format Modes	4-39
Text Job Format (Direct) Mode	4-40
The Status Structure	4-42
Pen Tables for Vector Devices	4-43
Constants and Data Types for Synonyms	4-45
General-Purpose PostScript Operator Synonym	4-45
PostScript Control Information Synonym	4-45
Dash Synonym	4-46
Halftone Synonym	4-46
Line Cap Synonym	4-47
Pattern Synonym	4-47
Cubic Synonym	4-48
QuickDraw Picture Synonym	4-49
Functions	4-49
Advanced Job Object Functions	4-50
Manipulating Printer Objects	4-54
Working With QuickDraw GX Print Files	4-61
Working With Paper Types	4-71
Formatting for Specific Devices	4-79
Color Profile Functions	4-84
Idle Job Function	4-90
Application-Defined Functions	4-90



Message Override Function for the Printing Status Dialog Box	4-90
Looping Through a Printer's View Devices	4-92
Looping Through a Job's Paper Types	4-92
The Status Resource	4-93
Summary of Advanced Printing Features	4-95

---

Glossary	GL-1
----------	------

---

Index	IN-1
-------	------

---



# Figures, Tables, and Listings

Preface

About This Book    xv

---

**Figure P-1**            Roadmap to the QuickDraw GX suite of books            xvi

Chapter 1

---

Introduction to Printing With QuickDraw GX    1-1

---

**Figure 1-1**            QuickDraw GX printing phases    1-4  
**Figure 1-2**            QuickDraw GX printing-related objects    1-7  
**Figure 1-3**            Dragging a document to a desktop printer icon    1-8  
**Figure 1-4**            Default QuickDraw GX desktop printer icons    1-9  
**Figure 1-5**            The Print dialog box    1-11  
**Figure 1-6**            The expanded Print dialog box    1-11  
**Figure 1-7**            The Print Time panel    1-12  
**Figure 1-8**            The Paper Match panel    1-12  
**Figure 1-9**            Message handlers in a message chain    1-14  
**Figure 1-10**            Overriding the `gxPrintingEvent` message    1-15  
**Figure 1-11**            Effect of specifying a shape in the `form` property of a format object    1-17  
**Figure 1-12**            A paper type for printing on letterhead paper    1-18  
**Figure 1-13**            QuickDraw GX printing-related objects    1-21  
**Figure 1-14**            Printing-related items in the File menu    1-25  
**Figure 1-15**            Manipulating the job object in response to user actions    1-27  
**Figure 1-16**            Printing a document containing multiple formats    1-29  
  
**Table 1-1**            QuickDraw GX printing-related objects    1-22

Chapter 2

---

Core Printing Features    2-1

---

**Figure 2-1**            Objects needed to implement core printing features    2-4  
**Figure 2-2**            The job object    2-6  
**Figure 2-3**            The format object    2-7  
**Figure 2-4**            The paper-type object    2-8  
**Figure 2-5**            The Page Setup dialog box    2-35  
**Figure 2-6**            The expanded Page Setup dialog box    2-36  
**Figure 2-7**            The Print dialog box    2-38  
**Figure 2-8**            The expanded Print dialog box    2-38  
  
**Listing 2-1**            Creating a job object for a printable document    2-12  
**Listing 2-2**            Polling for errors after individual functions    2-15  
**Listing 2-3**            Polling for errors after groups of functions    2-16  
**Listing 2-4**            Override function for the `gxPrintingEvent` message    2-19  
**Listing 2-5**            Using the `GXPrintPage` function to print a document    2-21  
**Listing 2-6**            Using the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions to print a document    2-23

<b>Listing 2-7</b>	Using the <code>GXFlattenJobToHdl</code> function to save a job object	2-25
<b>Listing 2-8</b>	Using the <code>GXFlattenJob</code> function to save a job object	2-28
<b>Listing 2-9</b>	Disposing of a job object when you close a document	2-29
<b>Listing 2-10</b>	Using the <code>GXUnflattenJobFromHdl</code> function to retrieve a job object	2-30
<b>Listing 2-11</b>	Using the <code>GXUnflattenJob</code> function to retrieve a job object	2-32
<b>Listing 2-12</b>	Using the <code>GXGetFormatJob</code> function to obtain a job object	2-33
<b>Listing 2-13</b>	Using the <code>GXGetFormatDimensions</code> function	2-34
<b>Listing 2-14</b>	Displaying the Page Setup dialog box	2-36
<b>Listing 2-15</b>	Displaying the Print dialog box	2-39
<b>Listing 2-16</b>	Responding to the Print Documents Apple event and specifying an output printer	2-40
<b>Listing 2-17</b>	Updating a job when receiving resume events	2-43
<b>Listing 2-18</b>	Converting a print record into a job object	2-45

## Chapter 3

## Page Formatting and Dialog Box Customization 3-1

---

<b>Figure 3-1</b>	The job collection	3-10
<b>Figure 3-2</b>	The format collection	3-12
<b>Figure 3-3</b>	The paper-type collection	3-14
<b>Figure 3-4</b>	A three page document and its corresponding job and format objects	3-15
<b>Figure 3-5</b>	Manipulating the format object in response to user actions	3-17
<b>Figure 3-6</b>	Scaling a format object	3-19
<b>Figure 3-7</b>	Using a form to format a page	3-20
<b>Figure 3-8</b>	The expanded Custom Page Setup dialog box with two panels	3-22
<b>Figure 3-9</b>	Print dialog box with default page range	3-35
<b>Figure 3-10</b>	Print dialog box with replacement page range	3-37
<b>Figure 3-11</b>	Print dialog box with customized page range	3-39
<b>Figure 3-12</b>	The Custom Page Setup dialog box	3-40
<b>Figure 3-13</b>	The expanded Custom Page Setup dialog box	3-40
<b>Figure 3-14</b>	A four-page document in which page two uses a unique format object	3-41
<b>Figure 3-15</b>	A four-page document in which pages 2 and 3 use the same format object	3-45
<b>Figure 3-16</b>	A four-page document in which pages 2 and 3 use unique formats objects	3-48
<b>Figure 3-17</b>	Moving a format object from one document to another	3-55
<b>Figure 3-18</b>	A three-page document and its corresponding job object, format objects, and paper-type objects	3-58
<b>Figure 3-19</b>	A panel added to the Custom Page Setup dialog box	3-70
<b>Figure 3-20</b>	Panel resource	3-127
<b>Figure 3-21</b>	Extended item list resource	3-128
<b>Figure 3-22</b>	Radio button items	3-129
<b>Figure 3-23</b>	Checkbox and pop-up menu items	3-130
<b>Figure 3-24</b>	Integer and real edit text items	3-131
<b>Figure 3-25</b>	String editable text items	3-132
<b>Table 3-1</b>	Functions that enable dialog box panels	3-23

<b>Table 3-2</b>	Functions that forward a dialog box message	3-24
<b>Listing 3-1</b>	A panel resource definition template	3-24
<b>Listing 3-2</b>	The extended item list resource definition template	3-26
<b>Listing 3-3</b>	Accessing copies information stored in a job collection	3-28
<b>Listing 3-4</b>	Modifying the job collection to implement the Print One Copy menu item	3-29
<b>Listing 3-5</b>	Replacing collection items	3-31
<b>Listing 3-6</b>	Setting up a default page range	3-33
<b>Listing 3-7</b>	Setting up a replacement page range	3-35
<b>Listing 3-8</b>	Setting up a customized page range	3-37
<b>Listing 3-9</b>	Creating a format object for a page in a document	3-42
<b>Listing 3-10</b>	Cloning a format object for two pages in a document	3-46
<b>Listing 3-11</b>	Disposing of a format object for a page in a document and creating a new one	3-49
<b>Listing 3-12</b>	Adding a form to a format object	3-51
<b>Listing 3-13</b>	Storing halftone information in a format collection	3-53
<b>Listing 3-14</b>	Moving a format object from one document to another	3-56
<b>Listing 3-15</b>	Obtaining a format object's mapping	3-57
<b>Listing 3-16</b>	Obtaining the paper-type object associated with a format object	3-59
<b>Listing 3-17</b>	Using the <code>GXForEachJobFormatDo</code> function	3-60
<b>Listing 3-18</b>	Obtaining scaling information on each format object	3-61
<b>Listing 3-19</b>	Saving the correspondence between format objects and document pages in a format collection	3-62
<b>Listing 3-20</b>	Filling the handle	3-63
<b>Listing 3-21</b>	Retrieving the correspondence between document pages and format objects from a format collection	3-65
<b>Listing 3-22</b>	Setting up a new panel	3-68
<b>Listing 3-23</b>	Sample panel resource	3-70
<b>Listing 3-24</b>	Sample item list resource	3-71
<b>Listing 3-25</b>	Sample 'CNTL' resource	3-72
<b>Listing 3-26</b>	Sample extended item list resource	3-72
<b>Listing 3-27</b>	Sample 'MENU' resource	3-73
<b>Listing 3-28</b>	Installing an override function for the <code>gxParsePageRange</code> message	3-74
<b>Listing 3-29</b>	Override function for the <code>gxParsePageRange</code> message	3-75

## Chapter 4

### Advanced Printing Features 4-1

---

<b>Figure 4-1</b>	The printer object	4-6
<b>Figure 4-2</b>	The print file object	4-10
<b>Figure 4-3</b>	The status resource	4-93
<b>Table 4-1</b>	Printer driver types	4-7
<b>Table 4-2</b>	QuickDraw GX printing synonyms	4-12
<b>Table 4-3</b>	Print job format modes	4-20
<b>Table 4-4</b>	Text job format mode query options	4-36
<b>Table 4-5</b>	Status type IDs	4-43
<b>Table 4-6</b>	The actions of the <code>GXSetPrinterProfile</code> function	4-87
<b>Table 4-7</b>	The actions of the <code>GXSetFormatProfile</code> function	4-89

<b>Table 4-8</b>	Status types	4-94
<b>Listing 4-1</b>	Obtaining the names and types of a printer and printer driver	4-22
<b>Listing 4-2</b>	Setting the job object's reference constant property	4-23
<b>Listing 4-3</b>	Getting the job object's reference constant property	4-24
<b>Listing 4-4</b>	Copying job object information	4-25
<b>Listing 4-5</b>	Determining a printer's resolution	4-26
<b>Listing 4-6</b>	Retrieving the printer's color profile and color space	4-27
<b>Listing 4-7</b>	Using the printer's color profile to convert colors	4-28
<b>Listing 4-8</b>	Opening and closing a print file	4-29
<b>Listing 4-9</b>	Reading a page from a print file	4-31
<b>Listing 4-10</b>	Creating a new paper-type object	4-32
<b>Listing 4-11</b>	Obtaining a paper-type object's name	4-32
<b>Listing 4-12</b>	Obtaining page and paper rectangles for a paper-type object	4-33
<b>Listing 4-13</b>	Executing a function for each paper-type object	4-34
<b>Listing 4-14</b>	Executing a procedure for each paper-type object	4-35

# About This Book

---

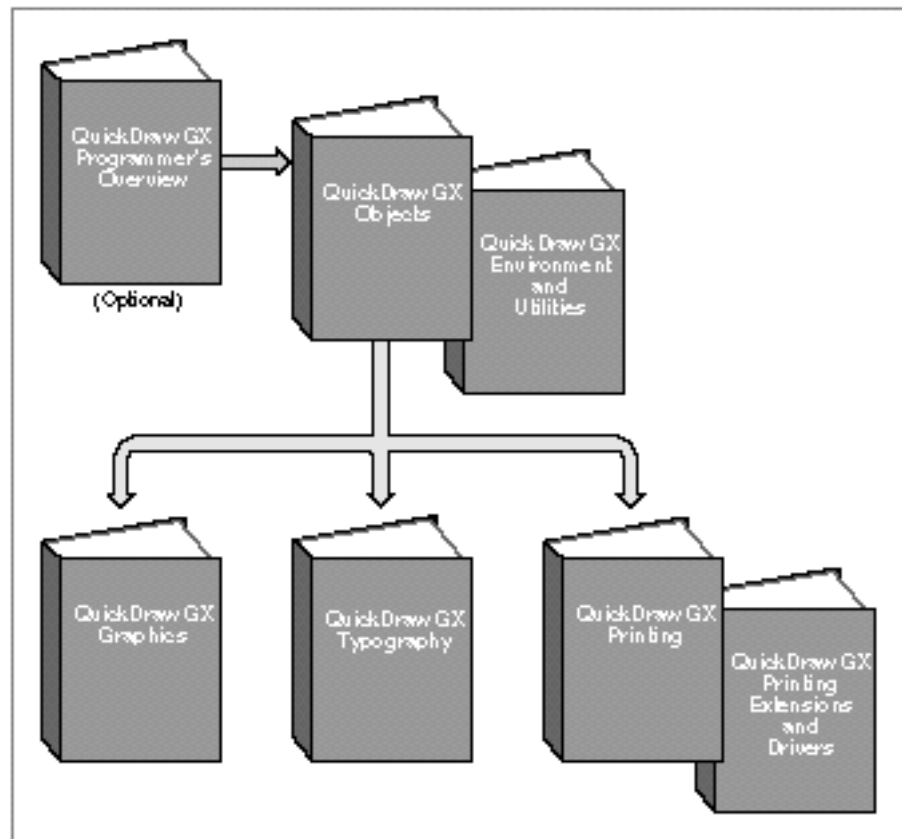
QuickDraw GX is an integrated, object-based approach to graphics programming on Macintosh computers. This book, *Inside Macintosh: QuickDraw GX Printing*, describes how to design your application to use the printing features of QuickDraw GX. It begins with an introduction to printing with QuickDraw GX and discusses architectural aspects of QuickDraw GX printing features—printing-related objects and the user interfaces. Then the book separates QuickDraw GX printing features into core features, page formatting and dialog box customization, and advanced features. You only need to read as many chapters as apply to your application's printing needs.

Before you begin this book, you should already be familiar with the QuickDraw GX environment and QuickDraw GX objects. An overview of the environment and objects is provided in the introductory chapter of *Inside Macintosh: QuickDraw GX Objects*. Complete information can be found in *Inside Macintosh: QuickDraw GX Environment and Utilities* and the other chapters of *Inside Macintosh: QuickDraw GX Objects*.

For more information about programming with QuickDraw GX, you need to refer to other books in the QuickDraw GX suite, including *Inside Macintosh: QuickDraw GX Objects*, *Inside Macintosh: QuickDraw GX Graphics*, and *Inside Macintosh: QuickDraw GX Typography*. If you need information on how to use QuickDraw GX to write printer drivers or printing extensions, see *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Figure P-1 shows the suggested reading order for the QuickDraw GX suite of books. A pictorial overview of *Inside Macintosh*, including the QuickDraw GX suite of books, appears inside the back cover.

**Figure P-1** Roadmap to the QuickDraw GX suite of books



## What to Read

This book is intended for developers who are interested in providing a QuickDraw GX printing capability in their applications. You can design your application to use the QuickDraw GX application-programming interface (API) for printing, even if the application doesn't use the graphics and typographic capabilities of QuickDraw GX.



In this book, each succeeding chapter builds on the previous chapter's information. So it's important to begin by learning the QuickDraw GX printing concepts and terms that are in Chapter 1, "Introduction to Printing With QuickDraw GX." This chapter presents an overview of printing with QuickDraw GX and briefly describes the dialog boxes that QuickDraw GX provides for user interaction with the printing process.

Most applications only need to support the set of printing features that are described in Chapter 2, "Core Printing Features." You use the core printing features when printing documents using QuickDraw GX. You also use them to display the standard printing-related dialog boxes and to print documents that were originally created to print with previous versions of the Macintosh printing architecture.

However, if you want to add panels to QuickDraw GX print dialog boxes to provide special features that require additional user specification, or if you want to manipulate the objects that QuickDraw GX uses to format the pages of a document, you also need to read Chapter 3, "Page Formatting and Dialog Box Customization." For example, through QuickDraw GX, your application can allow users to specify unique formats for the individual pages of a printable document.

Features that go beyond the core set and beyond those that allow you to handle page-by-page formatting and dialog box customization are described in Chapter 4, "Advanced Printing Features." You can use these features to optimize output for the capabilities of a particular device, create a file that is application-independent, define custom paper sizes, and more.

The first two pages of this book are color plates. Plate 1 shows an example of the QuickDraw GX color separation capability. Plate 2 shows common color-transfer modes used in printing.

## Chapter Organization

---

Most chapters in this book follow a standard general structure. For example, the chapter "Core Printing Features" contains these sections:

- n "About Core Printing Features." This section provides an overview of the core printing features provided by QuickDraw GX.
- n "Using Core Printing Features." This section describes the tasks you can accomplish using the core printing features of QuickDraw GX. It describes how to use the most common functions, gives related user interface information, provides code samples, and supplies additional information.

- n “Core Printing Features Reference.” This section provides a complete reference for the core printing calls by describing the data structures and functions you can use. Each function description follows a standard format, which gives the function declaration; a description of every parameter; the function result, if any; and a list of result codes. Most function descriptions give additional information about using the function and include cross-references to related information elsewhere.
- n “Summary of Core Printing Features.” This section shows the C interface for the constants, data types, and functions associated with the core printing features.

## Conventions Used in This Book

---

This book uses various conventions to present certain types of information.

### Special Fonts

---

All code listings, reserved words, and the names of data structures, constants, fields, parameters, and functions are shown in Courier (`this is Courier`).

When new terms are introduced, they are in **boldface**. These terms are also defined in the glossary.

### Types of Notes

---

There are several types of notes used in this book.

#### Note

A note like this contains information that is interesting but possibly not essential to an understanding of the main text. (An example appears on page 1-10.) u

#### IMPORTANT

A note like this contains information that is especially important. (An example appears on page 2-49.) s

### Numerical Formats

---

Hexadecimal numbers are shown in this format: 0x0008.

The numerical values of constants are shown in decimal, unless the constants are flag or mask elements that can be summed, in which case they are shown in hexadecimal.

## Type Definitions for Enumerations

---

Enumeration declarations in this book are commonly followed by a type definition that is not strictly part of the enumeration. You can use the type to specify one of the enumerated values for a parameter or field. The type name is usually the singular of the enumeration name, as in the following example:

```
enum gxDashAttributes {
    gxBendDash      = 0x0001,
    gxBreakDash     = 0x0002,
    gxClipDash      = 0x0004,
    gxLevelDash     = 0x0008,
    gxAutoAdvanceDash = 0x0010
};
typedef long gxDashAttribute;
```

## Illustrations

---

This book uses several conventions in its illustrations.

In illustrations that show object properties, properties that are object references are in *italics*. For example, see Figure 1-13 on page 1-21.

Objects in diagrams, whether shown with their properties or without, are represented by distinctive icons, such as these:



See, for example, Figure 1-2 on page 1-7.

## Development Environment

---

The QuickDraw GX functions described in this book are available using C interfaces. How you access these functions depends on the development environment you are using.

Code listings in this book are shown in ANSI C. They suggest methods of using various functions and illustrate techniques for accomplishing particular tasks. Although most code listings have been compiled and tested, Apple Computer, Inc., does not intend for you to use these code samples in your applications.

## For More Information

---

APDA is Apple's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

APDA

Apple Computer, Inc.

P.O. Box 319

Buffalo, NY 14207-0319

Telephone	1-800-282-2732 (United States) 1-800-637-0029 (Canada) 716-871-6555 (International)
-----------	---

Fax	716-871-6511
-----	--------------

AppleLink	APDA
-----------	------

America Online	APDAorder
----------------	-----------

CompuServe	76666,2405
------------	------------

Internet	APDA@applelink.apple.com
----------	--------------------------

If you provide commercial products and services, call 408-974-4897 for information on the developer support programs available from Apple.

# Introduction to Printing With QuickDraw GX

---

## Contents

About QuickDraw GX Printing	1-3
Core Printing-Related Objects	1-6
Desktop Printers	1-7
Print Files	1-8
Printer Drivers	1-8
Printing Extensions	1-9
Dialog Boxes	1-10
Message Passing	1-13
About QuickDraw GX Printing-Related Objects	1-16
Job Objects	1-16
Format Objects	1-17
Paper-Type Objects	1-18
Collection Objects	1-18
Printer Objects	1-20
Print File Objects	1-20
Summary of QuickDraw GX Printing-Related Objects	1-20
Using Printing-Related Objects With Other QuickDraw GX Objects	1-23
Shape Objects	1-23
Tag Objects	1-24
View Port Objects	1-24
View Device Objects	1-25
Implementing QuickDraw GX Printing Features	1-25
Core Printing Features	1-26
Customizing QuickDraw GX Printing Features	1-28
Advanced Printing Features	1-30
Compatibility With the Macintosh Printing Manager	1-30



This chapter introduces the primary features of printing with QuickDraw GX and gives you the overview you need to begin designing your application with printing in mind.

Before reading this chapter, you should be familiar with the general QuickDraw GX capabilities, and especially, you should be familiar with the use of objects. For an overview of QuickDraw GX and objects, see the introductory chapter of *Inside Macintosh: QuickDraw GX Objects*.

This chapter begins by showing how QuickDraw GX printing works and which phases of printing are of interest to the application developer. It also provides background information to set the stage for the remaining sections. This chapter then

- n introduces QuickDraw GX objects that directly support printing
- n describes how these printing-related objects are used with other QuickDraw GX objects
- n describes a strategy for implementing QuickDraw GX printing features
- n discusses compatibility between QuickDraw GX printing and the Macintosh Printing Manager

## About QuickDraw GX Printing

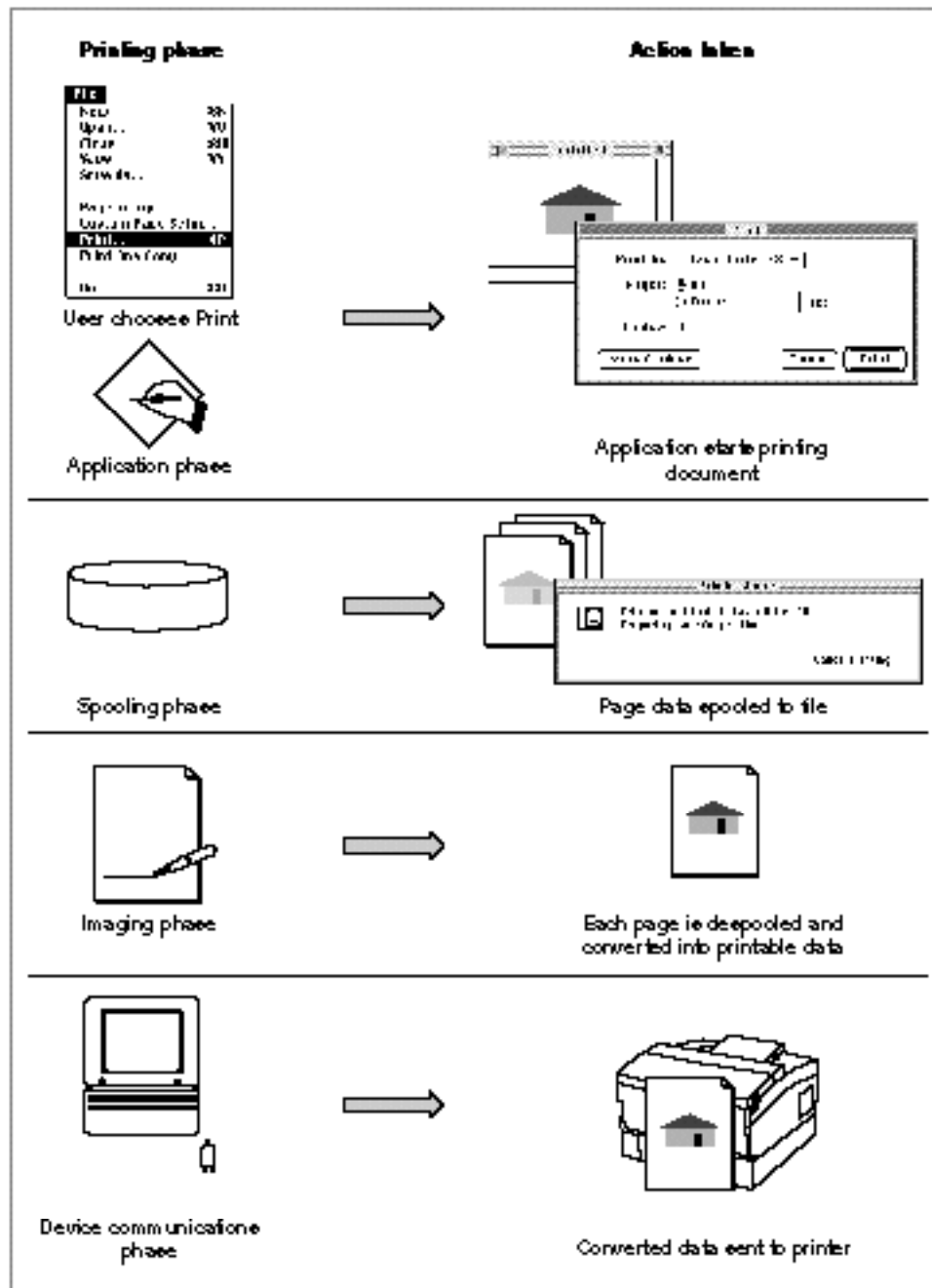
---

Printing with QuickDraw GX involves the interaction of your application program with components that QuickDraw GX provides, or components that may be provided by a printer manufacturer or other vendor. These components are

- n printer drivers that translate QuickDraw GX shapes into instructions for rendering the shapes on a device
- n printing extensions that provide additional capabilities for the printing system

QuickDraw GX actually performs most of the translation work itself so that the developer of a printer driver or printing extension can concentrate on the unique features or characteristics of a printing device. As an application developer, your work primarily consists of responding to printing-related menu selections and dialog boxes within the application.

To understand the division of labor, consider the model of QuickDraw GX printing phases in Figure 1-1.

**Figure 1-1** QuickDraw GX printing phases

There are four phases of printing:

- n The **application phase**, in which the application calls QuickDraw GX functions in response to the user choosing a menu item or changing an item in a dialog box. For example, when the user selects Print from the menu, the application calls functions to



display the Print dialog box and respond to the user's choices. One of the application's responses is to print the requested pages of a document, which leads to the spooling phase of printing.

- n The **spooling phase**, in which the requested pages are placed in a spool file. The application calls QuickDraw GX functions to perform this task, which is carried out collaboratively by QuickDraw GX, the printer driver, and any printing extensions that are active. From the application developer's point of view, it is seldom necessary to know how the work is divided between QuickDraw GX, a printer driver, or a printing extension. Thus, in this book, all collaborative efforts by these components are considered as being performed by the printer driver.
- n The **imaging phase**, in which the requested pages are despoiled by the printer driver and the contents are translated into instructions for the printer.
- n The **device communications phase**, in which the instructions are actually sent to the printer hardware.

As an application developer, you are primarily concerned with the application phase of printing. You may be interested indirectly in events in other phases because some of those events can be controlled by the application. For example, your application can provide alternative instructions for rendering output, rather than use the instructions generated by the printer driver. These alternative instructions are called **synonyms**. As another example, the application can retrieve and modify the contents of a file after it has been spooled.

This book provides all the information you need to implement QuickDraw GX printing in an application. For information about implementing printer drivers or printing extensions, see *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*. The following sections introduce topics that provide conceptual background for implementing QuickDraw GX printing features. The topics are

- n Core printing-related objects, which are objects that are used in every application and work together to support QuickDraw GX printing.
- n Desktop printers, which represent printers to the user as icons on the desktop.
- n Print files, which are the output of the spooling phase. A special kind of print file that can be opened and displayed without needing the fonts or application with which it was created is called a **portable digital document**, or PDD.
- n Printer drivers, which are responsible for defining the characteristics of the printing environment in addition to providing translation between the QuickDraw GX graphics representation of a page and the instructions that render it on a printer.
- n Printing extensions, which are add-on software that provide an additional level of customization to QuickDraw GX printing.
- n Dialog boxes, which are extensible in QuickDraw GX and, if extended, use additional resource types. Dialog boxes also require additional support because they are movable, requiring the screen behind them to be redrawn when they are moved.
- n Message passing, which is the basic technique used by the QuickDraw GX printing system to communicate between the application, printer driver, and printing extensions. It is also the technique used to notify the application when dialog boxes are moved.

## Core Printing-Related Objects

---

QuickDraw GX uses objects to represent printing-related data in the same way it uses objects in its other major components, graphics and typography. The core QuickDraw GX printing-related objects are job objects, format objects, and paper-type objects. There are other printing-related objects that provide additional information in support of the core objects or represent printers and files.

Because printing-related objects are interrelated, this section briefly describes how these objects work together to provide a complete specification for printing a document. For a more detailed description of each object, including those that support the core printing-related objects, see “About QuickDraw GX Printing-Related Objects” on page 1-16. The other chapters in this book provide a complete description of the printing-related objects and show how to use them.

In QuickDraw GX printing, a job object specifies everything QuickDraw GX needs to render a document. The most important specifications include the following ones; however, there are many others:

- n which pages to print
- n which printer is to receive the output
- n how to format the document; for example, for a particular page size and orientation, such as 8.5-by-11 inches and landscape

The pages to print and the printer on which to print them are typically straightforward specifications. The formatting specification can be more involved, however, because QuickDraw GX provides these formatting features:

- n You can print to a printer other than the one the document is formatted for; in other words, you can print without automatically reformatting the document.
- n You can specify a different format for each page of a document.

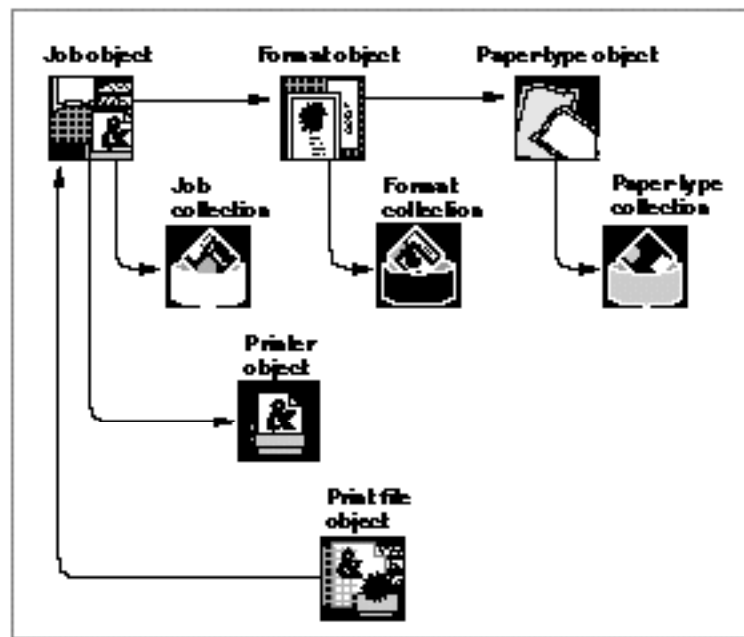
To support the first feature, the job object retains the formatting information for the formatting printer separately from the formatting information for the output printer. This allows a print job to be associated with two printers at the same time. The formatting printer specifies the document’s format. The output printer is the printer to which the document is sent to be printed. The document retains the format specified by the formatting printer even though the output printer may affect the appearance of the printed document. This feature is useful, for example, if you have formatted a long document for a typesetter but want to make a final check of a page or two on a StyleWriter printer.

To support the second feature, QuickDraw GX provides a format object. A format object can be specified for each page as it is printed. All pages can use the same format object, or selected pages can use different format objects. If desired, each page could use a different format object. You might also print the same page several times, each time specifying a different format object.

Associated with a format is a paper-type object. A paper-type object specifies the characteristics of the paper on which a page is printed. A paper-type object is separate from a format object because several format objects can share the same paper-type object.

Collection objects contain additional but less frequently used information about the job, formats, and paper types. The printer object represents a printer, and the print file object represents the spooled document or a portable digital document. Figure 1-2 shows the relationship between the core printing-related objects, the collection objects that support them, and the printer and print file objects.

**Figure 1-2** QuickDraw GX printing-related objects

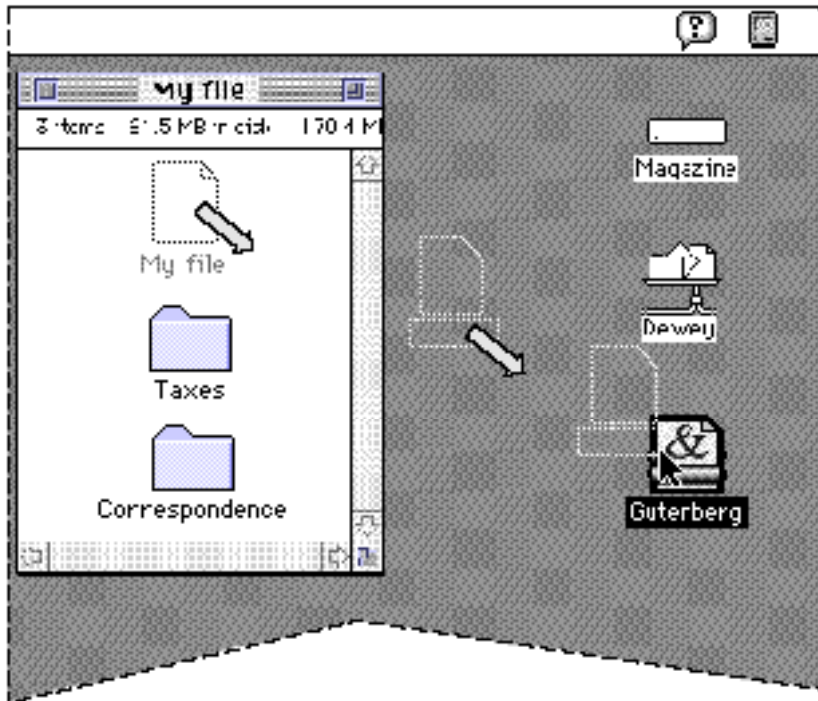


For more information about each object, see “About QuickDraw GX Printing-Related Objects” on page 1-16.

## Desktop Printers

In QuickDraw GX, a printer is represented by an icon on the desktop, which is similar to the way a hard disk or a shared volume is represented on the desktop. Thus, in QuickDraw GX, printers are often called **desktop printers**. A desktop printer is more than just an icon, however, because a desktop printer is associated with a queue to which print jobs are sent. A desktop printer also provides the ability to control the queue and to control the hardware itself with software.

A user can print from the Finder by dragging the document to the desktop printer icon. Figure 1-3 shows the user dragging the document “My file” to the desktop printer icon named “Gutenberg.” When the user releases the mouse, QuickDraw GX puts the print file representation of the document into the printer’s queue.

**Figure 1-3** Dragging a document to a desktop printer icon

Your application must implement the Print Document Apple event that allows Finder printing. For more information about Finder printing, see the chapter “Core Printing Features” in this book.

## Print Files

A **print file** is a document that has been spooled to a file through the printing process. The only way to create a print file is to print from the application, which causes the document’s contents to be spooled in a print file. If you wish, your application can retrieve a print file and insert, delete, or replace pages.

One kind of file an application might retrieve is a portable digital document, which is the kind of file that is created by selecting the PDD Maker GX desktop printer icon and then printing the document or by dragging the document to the icon. For an example of this icon, see Figure 1-4 on page 1-9.

## Printer Drivers

A QuickDraw GX **printer driver** defines the characteristics of a printer and the services the printer provides. The printer driver also translates QuickDraw GX shapes into instructions or operators that the printer understands, such as PostScript™. In reality,

much of a printer driver's standard functionality, such as PostScript conversion, is performed by QuickDraw GX for the printer driver.

From the application developer's point of view, it is useful to group printer driver-supplied features, printing extension-supplied features, and QuickDraw GX rendering features together because they are represented by the printer object. You can query the printer object for the characteristics of a printer, whether set by the printer driver, printing extension, or QuickDraw GX.

For example, your application can query the printer object to determine how best to print to the device that the printer object represents. Many of the default settings, such as page size and landscape or portrait orientation, are specified by the printer driver.

The printer driver is responsible for providing the printer icon to display on the desktop. Figure 1-4 shows examples of desktop printer icons for various devices.

**Figure 1-4** Default QuickDraw GX desktop printer icons



These devices need not represent actual physical devices on the system. In particular, the portable digital document printer driver, represented by the PDD Maker GX icon, is used only to create a document that is packaged ready-to-view on another computer.

## Printing Extensions

A QuickDraw GX **printing extension** defines add-on functionality that may be useful for several applications and whose usefulness is not restricted to a particular printer driver. For example, you may want a light-gray “Confidential” banner to appear as the backdrop on each printed page. Because several applications may need this kind of feature and these applications may print to a variety of printers, this kind of feature typically is implemented in a printing extension rather than as part of the application or printer driver.

**Note**

If you wish to provide functionality similar to a printing extension, such as a backdrop banner specific to your application, you can create a form shape and attach it to your format object. For an example of a form shape, see Figure 1-11 on page 1-17. For information about the form property of the format object, see the chapter “Page Formatting and Dialog Box Customization” in this book. u

## Dialog Boxes

---

QuickDraw GX print dialog boxes provide several key features:

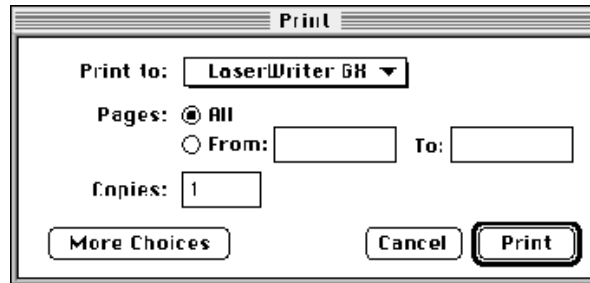
- n They are extensible, which allows you to collect or display information that is not in the default dialog boxes.
- n They are movable in addition to being modal. The ability to drag a dialog box around the screen overcomes some of the inconvenience of modality in that the user can move the dialog box if needed information in an underlying window is obscured. The user is allowed to switch to a different application while the dialog box is active, as well.
- n They can be set up to provide cut, copy, and paste editing operations.
- n The application’s response to user choices in a dialog box can be automated by specifying actions in resources associated with the dialog box; less procedural code is required in the application.

QuickDraw GX provides three kinds of print dialog boxes that you can access in your application:

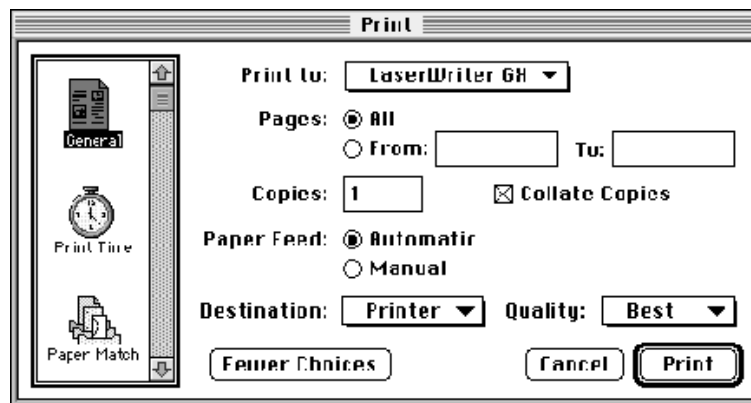
- n The Print dialog box appears in response to a request to print a document.
- n The Page Setup dialog box appears in response to a request to change the default formatting for the document.
- n The Custom Page Setup dialog box appears in response to a request to change the formatting of individual pages within a document.

In addition to these dialog boxes, the Printing Status dialog box appears when the application is spooling a document to a print file.

Most dialog boxes display in both a normal view and an expanded view. You use the normal view to display and accept the minimum amount of information that allows the user to conveniently proceed with the task. Figure 1-5 shows the default Print dialog box in its normal view.

**Figure 1-5** The Print dialog box

The expanded view displays the complete range of options. Figure 1-6 shows the expanded view of the Print dialog box.

**Figure 1-6** The expanded Print dialog box

Expanded views are divided into **panels**, which are subsets of the dialog box used to display and collect related pieces of information. You can add panels to a dialog box in the same way that a printer driver or printing extension may add panels. In Figure 1-6, the expanded view is currently displaying information in the General panel. Each panel is associated with an icon that displays in a scrolling list to the left of the panel. The name of the panel appears underneath its icon.

Figure 1-7 shows the Print Time panel. This panel allows a user to specify information related to a particular print job, such as the print job's priority and designated time to print.

**Figure 1-7** The Print Time panel

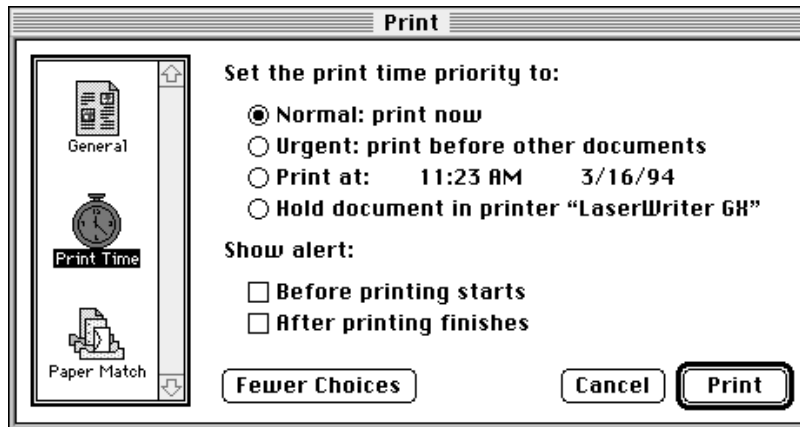
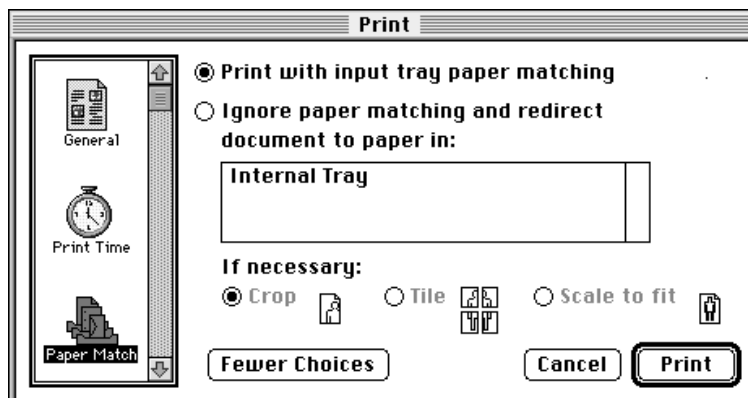


Figure 1-8 shows the Paper Match panel. This panel allows a user to specify information related to a print job's paper type, such as standard or special paper mapping.

**Figure 1-8** The Paper Match panel





You can use the following resources to add panels to dialog boxes:

Resource	Type	Description
Item list resource	'DITL'	Specifies a list of items in a dialog box, as described in the Dialog Manager chapter of <i>Inside Macintosh: Macintosh Toolbox Essentials</i> .
Panel resource	'ppnl'	Names a panel and associates it with an item list resource and an icon resource. For more information, see the chapter “Page Formatting and Dialog Box Customization” in this book.
Extended item list resource	'xdtl'	Specifies the actions to take when an item is manipulated; for example, when the user clicks a radio button. For more information, see the chapter “Page Formatting and Dialog Box Customization” in this book.

Additional resources may be needed. For example, many items in a dialog box are themselves defined as control or menu resources.

As mentioned previously, QuickDraw GX print dialog boxes are movable as well as being modal. When a user moves a dialog box, you are responsible for redrawing the screen that was behind it. QuickDraw GX notifies you that an update event occurred when this happens. The notification is provided by QuickDraw GX passing a message to the application, as described in the next section.

## Message Passing

---

QuickDraw GX printing features are based on a message-passing architecture. The messaging technology used with QuickDraw GX is described in the Message Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. This section provides you with a brief overview so that you can respond to messages passed to your application.

QuickDraw GX sends **printing messages** when certain printing-related tasks need to be accomplished or when certain printing-related conditions arise, such as when a print dialog box is displayed or the user moves the dialog box. A printing message is a value that QuickDraw GX passes down a chain of message handlers. A **message handler** is the recipient of a message and can include the application, the printer driver, and any printing extensions. The **message chain** consists of eligible message handlers.

The application can install itself as a message handler for particular messages. Typically, these messages relate to dialog boxes. The message handler specifies the code to execute when the message is received. This code is called an **override function** because it overrides the actions of the other message handlers by changing the behavior associated with the message.

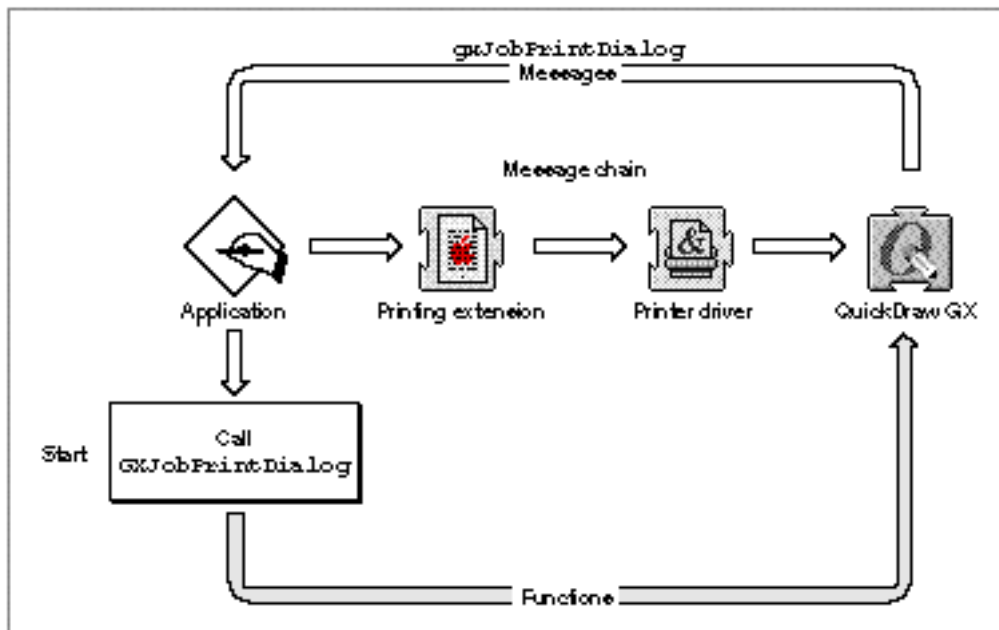
The override function can forward the message so that other message handlers can act on it. This situation is described as a **partial override**. If the function does not forward the message, the situation is described as a **total override** because other message handlers do not have a chance to act on the message.

If you are in doubt about whether to create a total override or not, try a partial override first because a total override may prevent an overlooked piece of code from being executed. For example, someone could provide a printing extension after your application has been distributed. The printing extension may rely on intercepting a message that was not previously required to be forwarded by your application and, thus, create an incompatibility between your application and the printing extension.

Two examples follow that show the typical cases in which an application needs to override QuickDraw GX messages. The first example shows how messages are involved in displaying a dialog box. The second example shows how a message is involved in handling movable dialog boxes.

When the user chooses the Print menu item, your application may wish to add a panel to the Print dialog box before it is displayed. Because you want the printer driver to provide the default dialog box, you install a message handler so you can override the Print dialog box message, `gxJobPrintDialog`. Figure 1-9 shows how this override happens with several message handlers in a message chain: the application, a printing extension, a printer driver, and QuickDraw GX.

**Figure 1-9** Message handlers in a message chain

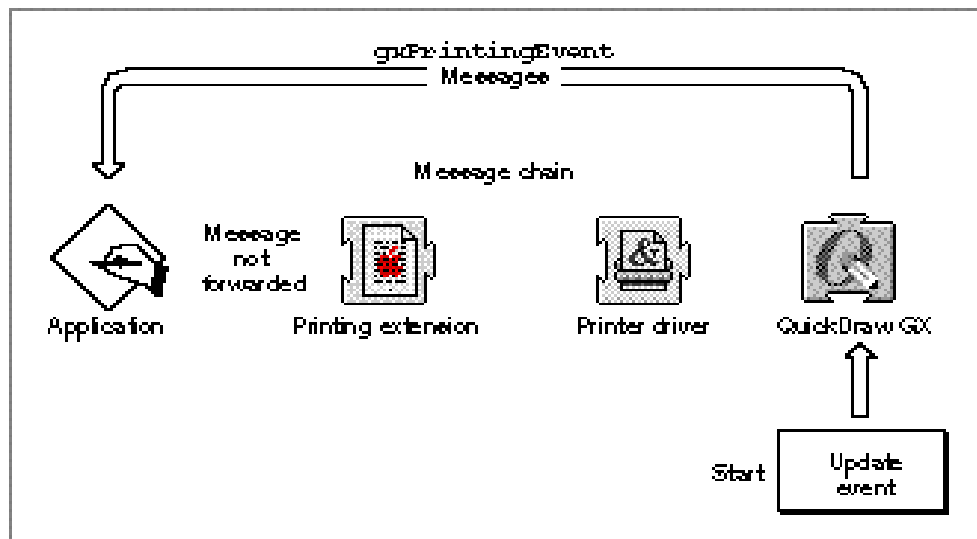


The application calls the `GXJobPrintDialog` function to display the dialog box. This function also causes QuickDraw GX to pass the `gxJobPrintDialog` message down the message chain, starting with the application. Because the application installed a function to respond to this message, the application's override function is called. (The override function is not shown in Figure 1-9.)

The override function is an application-defined function that is executed when QuickDraw GX sends the application the `gxJobPrintDialog` message. The override function adds a panel to the dialog box and forwards the message. By forwarding a message, each handler in the message chain—the application, printer driver, and printing extensions—participates in building the dialog box. The chapter “Page Formatting and Dialog Box Customization” in this book discusses the messages your application must override to add panels to QuickDraw GX movable modal dialog boxes.

Figure 1-10 shows an application that installs a function to be called when QuickDraw GX sends the `gxPrintingEvent` message. QuickDraw GX sends this message in response to an event, which allows the application to redraw a portion of the screen if the event is an update event caused by the user moving the dialog box.

**Figure 1-10** Overriding the `gxPrintingEvent` message



The override function that responds to the message has the responsibility to determine the kind of event that occurred and to redraw the invalid part of its windows if the event is an update event. This override function does not need to forward the message in this case because once the task is done, no other handler needs to take action. Thus, the function provides a total override of the `gxPrintingEvent` message in this case. For an example of an override function for this message, see the chapter “Core Printing Features” in this book.

## About QuickDraw GX Printing-Related Objects

---

The section “Core Printing-Related Objects” on page 1-6 describes how the job, format, and paper-type objects interrelate to define the printing environment for a document. The following sections describe each of the QuickDraw GX printing-related objects in more detail. At the end of these sections is a summary.

### Job Objects

---

The **job object** represents a print job that controls the way a document is printed. It contains properties to reference a formatting printer and an output printer. The **formatting printer** controls how the document is formatted. The **output printer** is the printer on which pages are printed. These specifications allow a document to be printed on the output printer yet retain the format specified by the formatting printer.

The job object also specifies additional properties, which include the following:

- n **Reference constant.** This property can be used for any application-specific purpose. For example, it can point to the contents of a document. It is discussed in the chapter “Advanced Printing Features” in this book.
- n **Error.** This property contains the last error associated with the print job. For information about accessing the error code, see the chapter “Core Printing Features” in this book.
- n **Format list.** This property specifies all of the formats that may be used with this print job. The first format in the list is the default format. For information about accessing the format list, see the chapter “Core Printing Features” in this book.
- n **Paper-type list.** This property specifies all of the paper types that may be used with this print job. The printer driver specifies the paper types in this list, although you can create new ones and add them to the list.
- n **Format mode.** This property specifies the preferred mode of printing the document associated with the print job; for example, using QuickDraw GX shape rendering, using raw PostScript, or using built-in fonts in the printer. For more information about the job format mode, see the chapter “Advanced Printing Features” in this book.
- n **Page range.** This property specifies the pages to print. For information about determining the page range, see the chapter “Core Printing Features” in this book.
- n **Panel dimensions.** This property specifies the size of panels in print dialog boxes, such as the Print and Page Setup dialog boxes. It is useful if you do not use the extended dialog item list resource to process events in dialog boxes and need to know where an event, such as a mouse-down event, occurred.

A job object also refers to a collection of items that can be specified for a print job. For more information about the job collection, see “Collection Objects” on page 1-18.

Most other objects refer to the job object. The reference allows the other objects to obtain information about the print job with which they are associated, and it especially allows them access to the reference constant property that points to application-specific information.

## Format Objects

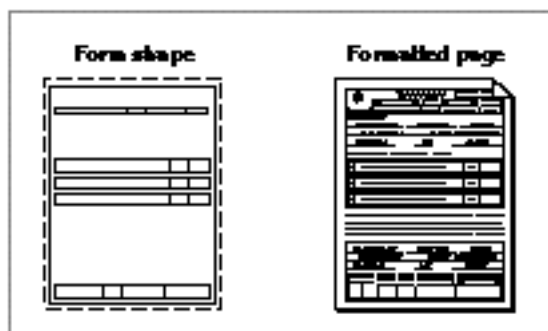
A **format object** specifies how a document or page of a document is to be formatted. The format object includes the following properties:

- n **Dimensions.** This property specifies size of the printable area. For more information about the dimensions property, see the chapter “Core Printing Features” in this book.
- n **Mapping.** This property determines the scale and orientation of the page. The mapping also determines the translation, skewing, and perspective as well; however, these are seldom changed. For more information about the mapping property, see the chapter “Page Formatting and Dialog Box Customization” in this book.
- n **Form.** This property specifies a shape object to print as a backdrop on each page of output and a mask shape that defines erasable areas within the form. For example, a form shape may provide a template so that each page of the document appears as if it is positioned within the template, or the form shape may appear as a logo or banner behind the contents of a page. For more information about form shapes, see the chapter “Page Formatting and Dialog Box Customization” in this book.
- n **Paper-type.** This property contains a reference to the paper-type object associated with this format. Because the paper-type object can restrict the printable area, you can use a paper-type object to change the printable area. For more information about the paper-type property, see the chapter “Advanced Printing Features” in this book.

A format object also refers to a collection of items that can be specified for a format. For more information about the format collection, see “Collection Objects” on page 1-18.

In the format object itself, you may change only the form property and the paper types. Figure 1-11 shows a form shape and how it can be used to format a document.

**Figure 1-11** Effect of specifying a shape in the form property of a format object



## Paper-Type Objects

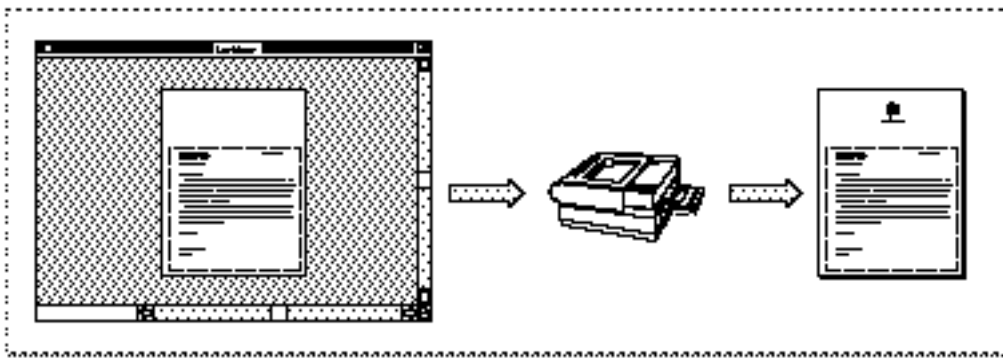
A **paper-type object** defines a paper type for a format. The paper-type object includes the following properties:

- n **Name.** This property specifies the name of the paper type. This name can be used to allow the user to select a paper type in a dialog box and is also used for paper matching.
- n **Dimensions.** This property specifies the size of the paper and the size of the printable area within the paper. This property allows you to specify a printable area that is different from the area specified by the dimensions property of the format object.

A paper-type object also refers to a collection of items that can be specified for a paper type. For more information about the paper-type collection, see the next section, “Collection Objects.”

Figure 1-12 shows an example of a paper type that restricts the printable area for printing on letterhead paper.

**Figure 1-12** A paper type for printing on letterhead paper



Paper-type objects are introduced in the chapter “Core Printing Features” in this book. Their use in defining different paper sizes is described in the chapter “Advanced Printing Features” in this book.

## Collection Objects

Collection objects are repositories for additional information associated with the core printing-related objects. Each piece of information is called an item. The print collection objects are

- n The **job collection**, which contains items of information that are relevant to a print job. These items include information about how to print the document; for example, how many copies, how to collate them, paper feed options, whether the document is to be printed to disk, and file information.

- n The **format collection**, which contains items of information related to printing a page from the document. It specifies the orientation of a page, whether a halftone should be applied, the scale, and other items related to formatting a page.
- n The **paper-type collection**, which contains items of information related to the kind of paper to which the format applies. For example, it specifies the base paper type, such as US letter or legal, and the units in which the paper is measured, such as inches or millimeters.

Figure 1-13 on page 1-21 shows the items that QuickDraw GX defines for these collection objects. They are discussed completely in the chapter “Page Formatting and Dialog Box Customization” in this book.

Typically, an item in a collection object is set by the printer driver. The user can change the item by setting values or controls in a dialog box. For example, the value in the copies information item of the job collection is set by the printer driver. The default Print dialog box allows the user to change the value. The value in the item is then used by the printer driver to determine how many times to print the pages associated with the job object.

You only need to be concerned about the information in collection objects in the following situations:

- n when you are printing without dialog boxes and need to set an item in a collection object
- n when you want to allow the user access to an item that is not provided by a printer driver in a dialog box

For an example of the first situation, to implement the Print One Copy menu item, you need to set the copies item in the job collection to 1 before printing and reset it to its previous value afterwards.

Consider the following example that applies to the second situation. The job object specifies the pages to print, which the printer driver uses, by default, in its Print dialog box. The job collection object provides a page-range information item that allows a complex range of pages to be specified. To support the complex page range, you must customize the Print dialog box to display the range from the collection item and store the new values back in the collection object when the user changes them. Of course, the printer driver must be set up to use the collection item too.

A printer driver can define additional items and store them in the appropriate collection. Your application can do likewise. You should consider whether these collections are appropriate for the kind of information you wish to manage. You can also create your own special-purpose collections. For more information about collections, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Printer Objects

---

A **printer object** represents the characteristics of a printer. They are set by the printer driver. You can determine these characteristics by referring to the output or formatting printer in a job object. You cannot change these characteristics.

The printer object includes the following properties:

- n **Printer name.** This property contains the printer's name; for example "All Mine."
- n **Printer type.** This property contains the kind of printer; for example, 'lwscl' for the LaserWriter II SC.
- n **Printer driver name.** This property contains the printer driver's name; for example, "LaserWriter II SC."
- n **Printer driver type.** This property contains the kind of printer driver; for example, 'post' for a Postscript printer.
- n **View device list.** This property refers to view devices that define a printer's resolution (dots-per-inch) and color space.

For information about each of these properties, see the chapter "Advanced Printing Features" in this book.

## Print File Objects

---

A **print file object** represents a file that is created by QuickDraw GX as the data is spooled to disk for printing. A special kind of print file is the portable digital document, which is created by the PDD Maker GX printer driver.

A print file is self-contained. When you open it, you specify a job object that QuickDraw GX sets up to match the characteristics of the job that printed the file. Thus, a print file retains information about the output and formatting printers, its format, paper types, and so on.

The print file object contains the following properties:

- n **Page count.** This property specifies the number of pages in the file.
- n **Format list.** This property specifies the format object for each page.
- n **Shape list.** This property specifies the picture shape object associated with each page.

For information about each of these properties, see the chapter "Advanced Printing Features" in this book.

## Summary of QuickDraw GX Printing-Related Objects

---

Figure 1-13 shows all the QuickDraw GX printing-related objects and collection objects. In this figure, references are represented by arrows. References to job objects, however, are not shown. Note that these are objects, not structures. The order of the contents is arbitrary. You access the contents procedurally by calling functions, not by accessing fields in a data structure.



Figure 1-13 QuickDraw GX printing-related objects

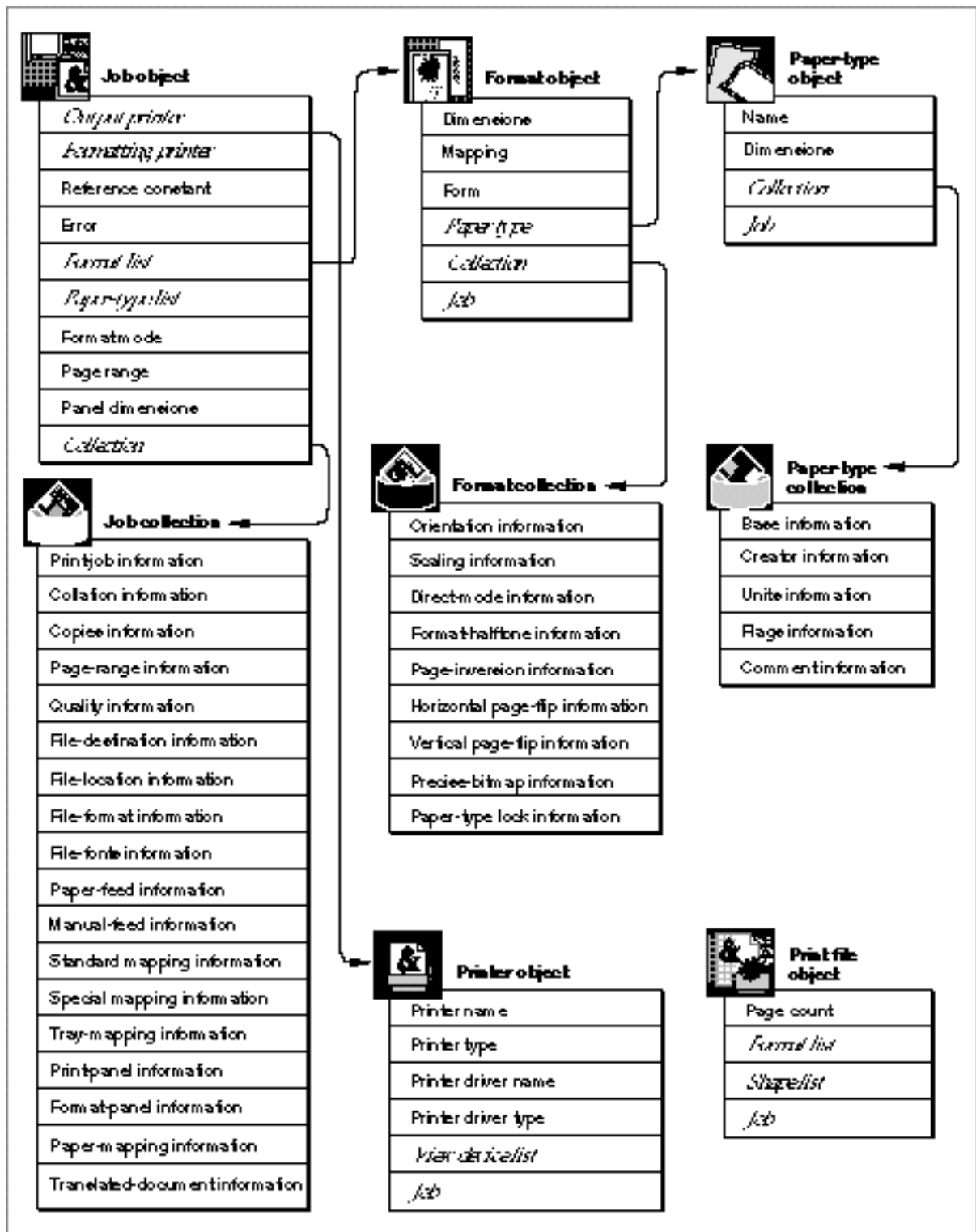


Table 1-1 describes the printing-related objects.

**Table 1-1** QuickDraw GX printing-related objects

Printing-related object	Description
Job	Holds the primary printing information for a document. Every printable document has a job object associated with it. The job object specifies the number of copies and a page range and includes references to one or more format objects and two printer objects.
Format	Specifies page-formatting characteristics such as scaling and page dimensions and includes a reference to a paper-type object.
Paper type	Specifies a paper-type name (such as “US Letter”), the physical dimensions of the paper, and the printable area within it.
Printer	Represents the capabilities of a physical printer and includes a name and type, a printer driver name and type, and a reference to one or more view device objects from which you can retrieve information about the printer’s characteristics.
Print file	Represents the file that results from printing, such as a spool file or a portable digital document.
Job collection	Contains items of information that are relevant to a print job. These items include information about how to print the document; for example, how many copies, how to collate them, paper feed options, whether the file is to be printed to disk, and file information.
Format collection	Contains items of information related to printing a page from the document. It specifies the dimensions for the page, the orientation, whether a halftone should be applied, the scale, and other items related to formatting a page.
Paper-type collection	Contains items of information related to the kind of paper to which the format applies. For example, it specifies the base paper type, such as US letter or legal, and the units in which the paper is measured, such as inches or millimeters.

## Using Printing-Related Objects With Other QuickDraw GX Objects

---

QuickDraw GX printing-related objects serve only one purpose—to support printing. The parts of your application unrelated to printing do not require the use of or access to printing-related objects. The parts of your application that do support printing, however, require the use of other QuickDraw GX objects. These objects include

- n shapes
- n tags
- n view ports
- n view devices

The use of these objects to support QuickDraw GX printing is well structured. The following sections discuss how these objects are used in QuickDraw GX printing.

### Shape Objects

---

Shape objects specify the content of what you want to render on a page of output. The format object, for example, allows you to specify a shape to be printed as a backdrop to the document's contents.

The document's contents are also represented as shapes. For example, text is typically represented as glyph or layout shapes. Graphics are specified by graphics shapes, such as lines, rectangles, polygons, paths, and so on. QuickDraw GX represents each page of output as a picture shape that contains these other shapes.

Either you can create a picture shape that represents the contents of the entire page, or you can allow QuickDraw GX to collect into a page the shapes you specify. For example, if you choose to create a picture shape and print it as a page, you pass the picture shape to the `GXPrintPage` function, which spools the page to the printer.

If you choose to specify individual shapes to be included in the page, you call the `GXStartPage` function to start building a picture shape and call the `GXDrawShape` function for each shape you want to render. When you call the `GXFinishPage` function, QuickDraw GX spools the picture shape for the page.

For an example of each way of printing using shape objects, see the chapter “Core Printing Features” in this book.

## Tag Objects

---

QuickDraw GX allows you to directly control the way that printing is performed through the use of synonyms stored in tag objects. You specify the action to take in a tag object and attach it to another object, such as a shape, ink, or transform. Here are two of the uses of tag objects:

- n Halftone specifications can be placed in a tag that is referred to by a shape's ink object. When the shape is drawn, QuickDraw GX draws it with the specified ink using the halftone in the ink's tag object.
- n PostScript operators can be placed in a tag that is referred to by the shape itself or by its style, ink, or transform objects. When the shape is drawn, the PostScript operators are used directly, in place of QuickDraw GX data.

For information about how to set up and attach tag objects to shapes, see the tag objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

## View Port Objects

---

View ports are used to restrict the parts of shapes that are spooled during printing. They also specify how to associate a shape with a view port when reading the shapes from a print file. For example, when you call the `GXStartPage` function to build your picture shape of the page, you specify a view port list. This view port list controls which shapes are printed. When you call the `GXDrawShape` function for a shape in order to add the shape to the picture shape, only the part of the shape that shows through a view port in this list is added to the picture shape. When a print file is read, the picture shape is associated with the view ports in the list you specify.

### Note

The `GXDrawShape` function may also cause the shape to be drawn onscreen. If you draw a shape with view ports that are in the onscreen view group but not specified in the view port list when calling the `GXStartPage` function, the shape is displayed on the screen. u

For more information about using view ports with the `GXStartPage` function, see the chapter “Core Printing Features” in this book. For more information about using view ports when reading print files, see the chapter “Advanced Printing Features” in this book.

## View Device Objects

---

Printer objects refer to view devices that are created by the printer driver. You can examine a printer's view devices to determine its characteristics, such as resolution, color set, and color profile. You cannot change these characteristics. For more information about accessing a printer object's view devices, see the chapter "Advanced Printing Features" in this book.

## Implementing QuickDraw GX Printing Features

---

As you prepare to implement QuickDraw GX printing, you need to consider which printing-related services your application will provide and what features QuickDraw GX provides to implement your services. Typically, the user expects to control printing through menus and dialog boxes in the application or by printing from the Finder. These are the core printing features that every application needs to implement. Figure 1-14 shows the File menu of a typical application that contains the printing-related menu items.

---

**Figure 1-14**     Printing-related items in the File menu

File	
New	⌘N
Open...	⌘O
Close	⌘W
Save	⌘S
Save As...	
Page Setup...	
Custom Page Setup...	
Print...	⌘P
Print One Copy	
Quit	⌘Q

As a core feature, of course, you allow the user to print the document. You also allow the user to format all the pages in a document the same way. The user chooses the Page Setup menu item to specify document formatting.

You may allow the user to customize the format of individual pages using the Custom Page Setup menu item. You may also wish to change the content of the dialog boxes from the defaults provided by the printer driver, printing extensions, and QuickDraw GX. You are implementing customization features when you provide page-by-page formatting and dialog box customization.

Other features that you may provide, but are probably not necessary to implement in most applications, are considered advanced printing features. Advanced printing features are not necessarily harder to implement than other features; it just is less likely that your application needs to provide them.

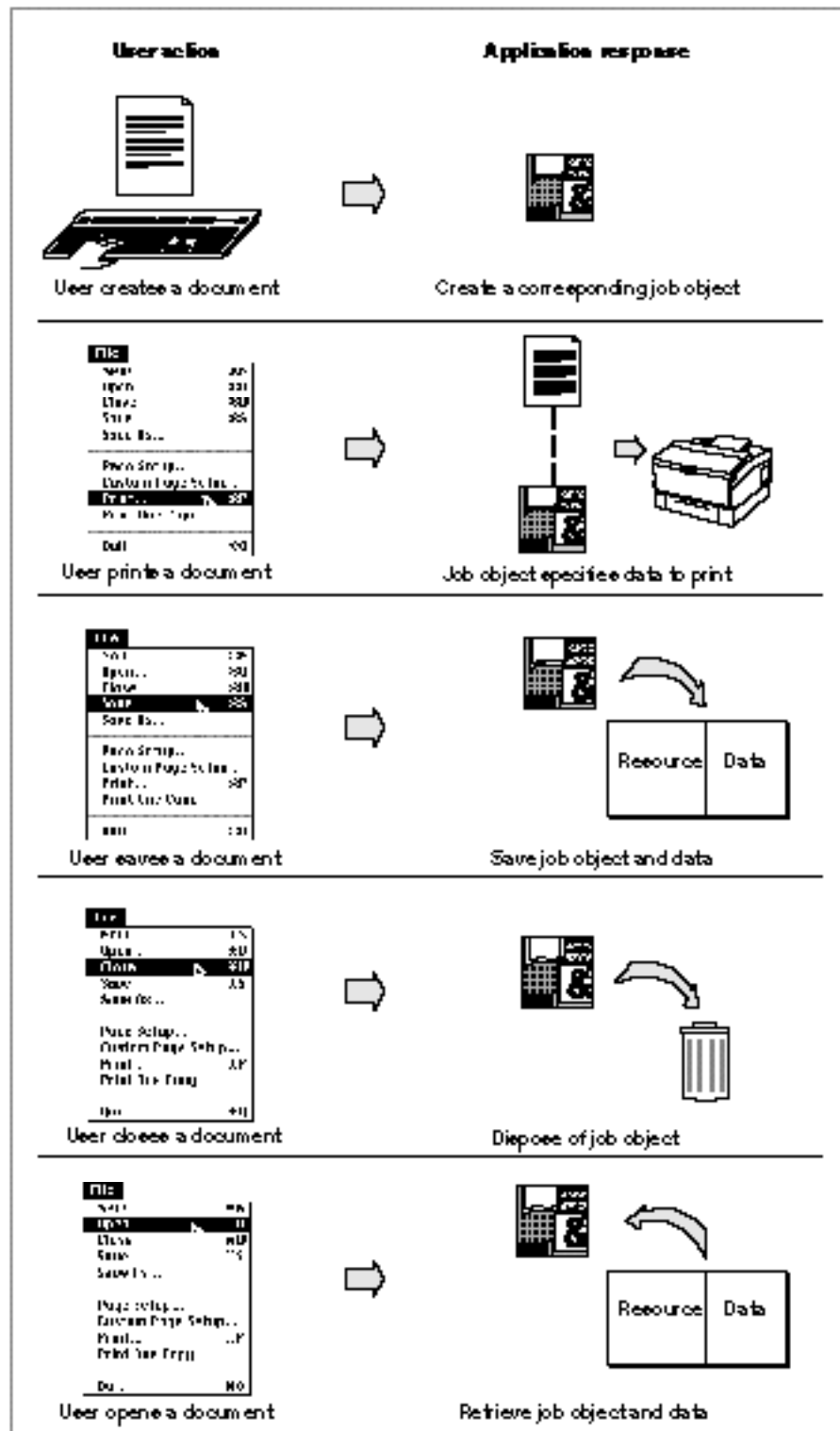
The following sections describe three classes of printing features:

- n core printing features
- n customization features
- n advanced printing features

## Core Printing Features

---

Generally, you work with printing-related objects when a user creates, saves, closes, or opens a printable document. A job object represents the primary association between a document, which is application-defined, and QuickDraw GX printing features. The job object represents a **print job** in the sense that it specifies the parameters for printing a document. Thus, core printing features require you to manipulate the job object. Figure 1-15 shows how you manipulate the job object in response to user actions.

**Figure 1-15** Manipulating the job object in response to user actions

When you create a job object, QuickDraw GX automatically provides you with default format and paper-type objects. The initial values of these objects are determined by the output printer that is currently selected when the job is created. These values can change if the user later changes the output printer.

You associate your document's data with the job object. QuickDraw GX maintains the relationship between the job, format, paper-type objects, and their collections. This is useful when you save or open a document because these objects must be flattened or unflattened, respectively.

Flattening and unflattening QuickDraw GX printing-related objects is very similar to flattening and unflattening a shape object. When you flatten a shape object, the style, ink, and transform objects are flattened with it. For printing-related objects, QuickDraw GX flattens all related objects with the job object, including multiple format, paper-type, and collection objects. They may be flattened in the form of a handle, which is convenient for writing the objects to the resource fork, or you can use your own procedure to store the job object and related objects wherever you wish.

You are responsible for displaying the Print and Page Setup dialog boxes. Because these dialog boxes are movable, your application must install a handler for the `gxPrintingEvent` message to update the screen if a dialog box moves.

Actual printing, which from the application's perspective means spooling the document to the printer driver, involves looping through the pages to be printed, and perhaps looping through the shapes to be included on each page. The work of applying formatting instructions and such is the responsibility of the printer driver.

There are several other things you must do to implement core printing features:

- Identify the location of the Edit menu and its items to allow QuickDraw GX to support the Cut, Copy, and Paste menu items when a print dialog box is active.
- Support printing from the Finder, which requires that your application support the Print Documents ( 'pdoc' ) Apple event and support this Apple event's optional attribute to allow the user to drag a copy of the document to a desktop printer for printing.
- Allow users to print documents originally created to print with the Macintosh Printing Manager.

None of these tasks are conceptually difficult. The chapter "Core Printing Features" in this book shows you how to perform each of these tasks.

## Customizing QuickDraw GX Printing Features

---

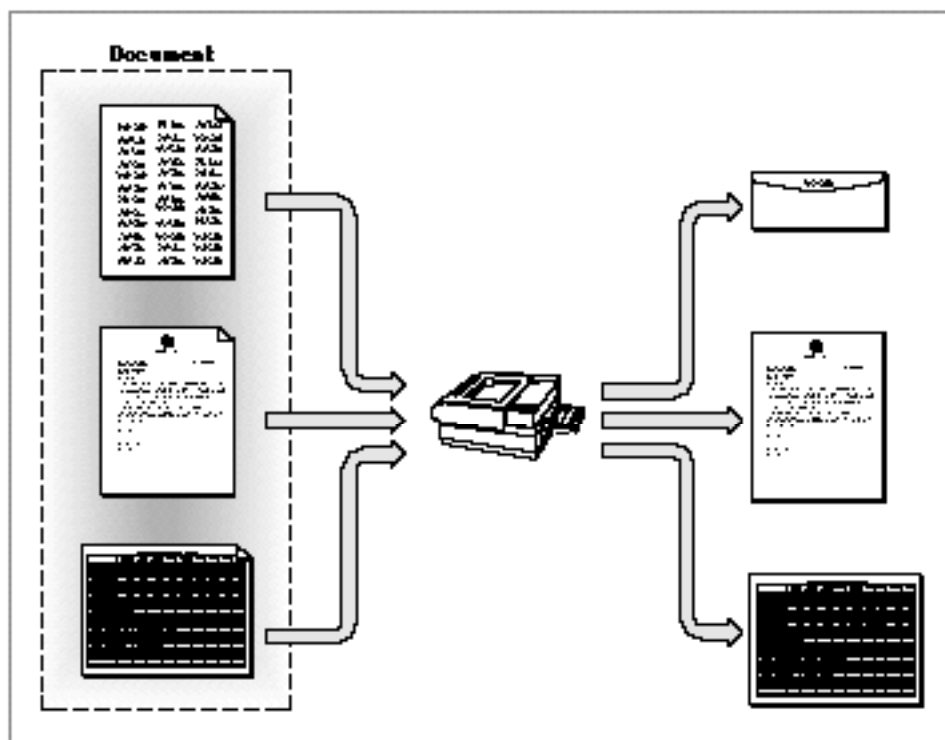
QuickDraw GX allows you to customize some of its features to address the needs of your particular application. If you want to manipulate the objects that QuickDraw GX uses to format the pages of a document or if you want to add panels to QuickDraw GX print dialog boxes, you need to read the chapter "Page Formatting and Dialog Box Customization" in this book.

Through QuickDraw GX, your application can allow users to specify unique formats for the individual pages of a printable document. For example, using QuickDraw GX, your



application can allow a user to create and print a single document that consists of an address page on an envelope, a business letter on a sheet of paper in portrait orientation, and a spreadsheet on a sheet of paper in landscape orientation. Figure 1-16 shows an example.

**Figure 1-16** Printing a document containing multiple formats



In addition, QuickDraw GX allows you to add panels to its dialog boxes to provide special features that require additional user specification. A **panel** is a portion of a dialog box in which an application can provide additional options for users. These specifications are stored as items in collection objects. For example, your application may add a panel that provides special color features, such as color separation and color choices or halftone information, which need to be stored with a job or format. QuickDraw GX dialog boxes are introduced in “Dialog Boxes,” which begins on page 1-10. For information about collections, see “Collection Objects” on page 1-18.

## Advanced Printing Features

---

QuickDraw GX provides several features that allow your application to provide additional control for users and allows the application to take advantage of features in particular printers. These features allow you to

- n provide access to and perhaps modify the contents of a portable digital document or other print file
- n use different paper-type objects, including those created with the PaperType Editor
- n take advantage of a printer's built in features, such as fast text-streaming using built-in fonts by way of a **direct job-formatting mode**
- n directly specify methods of rendering data with alternative representations of QuickDraw GX graphics objects, such as with raw PostScript (These alternative representations are called **synonyms**, which are stored in tag objects. For a brief introduction of how you implement synonyms, see "Tag Objects" on page 1-24.)
- n set up halftones on a shape-by-shape basis by specifying halftones for the inks they use
- n provide users with feedback about vector device capabilities
- n examine the characteristics of a printer, such as its resolution and color-rendering capabilities
- n change the job properties if the user switched printers
- n change or prevent the display of the Status dialog box

The chapter "Advanced Printing Features" in this book describes each of these features.

## Compatibility With the Macintosh Printing Manager

---

Non-QuickDraw GX versions of Macintosh system software use the Printing Manager, which QuickDraw GX replaces. The Printing Manager encompasses several concepts for which QuickDraw GX printing introduces parallel vocabulary. Old and new printing architecture terms include the following:

Printing Manager term	QuickDraw GX term
Printer driver	Printer driver and printing extensions
System printer	Default desktop printer
Print record	Job object
Spool file	Print file

To enable the printing of QuickDraw documents on QuickDraw GX printers, you must convert the document with the QuickDraw GX Translator, which is described in the environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*, and convert the print record by calling the `GXConvertPrintRecord` function, which is described in the chapter "Core Printing Features" in this book.

## CHAPTER 1

### Introduction to Printing With QuickDraw GX

#### **Note**

Printer drivers created with different versions of the Macintosh printing architecture can be present in a computer along with QuickDraw GX printer drivers. If QuickDraw GX is installed, the QuickDraw GX printer drivers are active; otherwise, the QuickDraw GX printer drivers are not active. u



# Core Printing Features

---

## Contents

About Core Printing Features	2-3
Core Print Objects	2-5
Job Object Properties	2-5
Format Object Properties	2-7
Paper-Type Object Properties	2-8
Edit Menu Structure	2-9
Using Core Printing Features	2-10
Initializing QuickDraw GX Printing	2-11
Creating a Job Object for a Printable Document	2-12
Error Handling	2-14
Supporting QuickDraw GX Print Dialog Boxes	2-17
Printing Documents Using QuickDraw GX	2-20
Printing Pages as Single Picture Shapes	2-21
Printing Pages by Capturing Shapes	2-22
Saving a Job Object With a Document File	2-24
Saving a Job Object in a Single Handle	2-25
Saving a Job Object Using a Flattening Function	2-27
Disposing of a Job Object When Closing a Document	2-28
Retrieving a Job Object When Opening a Document	2-29
Retrieving a Job Object From a Handle	2-30
Retrieving a Job Object Using an Unflattening Function	2-32
Obtaining Object References	2-33
Obtaining Information From a Format Object	2-33
Displaying QuickDraw GX Print Dialog Boxes	2-35
Displaying the Page Setup Dialog Box	2-35
Displaying the Print Dialog Box	2-37
Supporting Printing From the Finder	2-39
Updating Job Object Information	2-42
Printing Macintosh Printing Manager Documents	2-44

Core Printing Features Reference	2-46
Constants and Data Types	2-46
Gestalt Selectors for Printing	2-47
QuickDraw GX Printing-Related Objects	2-47
Edit Menu Location	2-48
Dialog Box Results	2-48
Functions	2-49
Initializing and Terminating QuickDraw GX Printing Features	2-50
GXInitPrinting	2-50
GXExitPrinting	2-51
Handling Errors	2-52
GXGetJobError	2-52
GXSetJobError	2-53
Creating and Managing Job Objects	2-54
GXNewJob	2-54
GXDisposeJob	2-55
GXFlattenJobToHdl	2-56
GXFlattenJob	2-57
GXUnflattenJobFromHdl	2-58
GXUnflattenJob	2-59
GXUpdateJob	2-60
Printing With QuickDraw GX	2-61
GXSelectJobOutputPrinter	2-61
GXGetJobPageRange	2-62
GXStartJob	2-63
GXPrintPage	2-64
GXFinishJob	2-65
GXStartPage	2-66
GXFinishPage	2-67
Obtaining Information on Printing-Related Objects	2-68
GXGetJobFormat	2-69
GXGetFormatJob	2-69
GXGetFormatDimensions	2-70
Displaying the Page Setup and Print Dialog Boxes	2-71
GXInstallApplicationOverride	2-71
GXJobDefaultFormatDialog	2-72
GXJobPrintDialog	2-73
Converting a Print Record	2-75
GXConvertPrintRecord	2-75
Application-Defined Functions	2-76
Message Override Functions	2-76
GXPrintingEvent	2-76
Flattening and Unflattening Functions for Job Objects	2-77
MyFlattenFunction	2-77
MyUnflattenFunction	2-78
Summary of Core Printing Features	2-79

## Core Printing Features

This chapter describes how your application can use the core set of QuickDraw GX printing features to print documents created with QuickDraw GX. Read the information in this chapter if you want to print your application's documents to an output device. For example, you might use QuickDraw GX to print to a LaserWriter a document that contains some text and a few illustrations.

Before reading this chapter, you should be familiar with the basic concepts and user interface for printing with QuickDraw GX, as described in the chapter "Introduction to Printing With QuickDraw GX" in this book.

This chapter describes the basic QuickDraw GX print objects: a job, a format, and a paper type. This chapter also shows you how to

- n set up the QuickDraw GX printing environment
- n create a job object that contains the information needed to print a document
- n detect error conditions
- n print your application's documents
- n save job object information when a user saves a document
- n dispose of a job object when a user closes a document
- n retrieve job object information when a user opens a document
- n obtain information on a format object
- n display QuickDraw GX print dialog boxes
- n support printing from the Finder
- n convert a print record into a job object to print existing documents designed for printing with the Macintosh Printing Manager

## About Core Printing Features

---

Core printing features are features that must be implemented to allow printing documents that contain QuickDraw GX graphics or typographical shapes. These features include the ability to print to desktop printers, format a document for a particular printer (a formatting printer), yet allow printing to another printer (the output printer) without reformatting the document. Core features also include the ability to print from the Finder and to print existing documents designed for printing with the Macintosh Printing Manager.

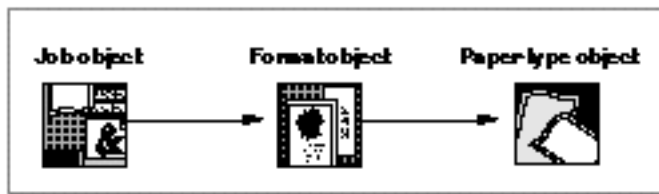
## Core Printing Features

To enable these core features, your application must manipulate three kinds of objects:

- n the **job object**, which contains information about the print job used to print a document
- n the **format object**, which contains information about how to format one or more pages of a document for printing
- n the **paper-type object**, which contains information about the paper on which a document is to be printed

Figure 2-1 shows the relationship between these objects.

**Figure 2-1** Objects needed to implement core printing features



All aspects of printing with QuickDraw GX relate to a particular job object. The job object defines the parameters with which to print the document, which a user typically specifies in the Print dialog box.

Your application sets up the correspondence between a document and a job object. The job object is tied to the format and paper-type objects through references. A job object refers to at least one format object. The format object specifies how to format the pages in a document. To implement core printing features, in which each page of a document is formatted the same way, you are only concerned about the first reference to a format object because this format object represents the default format.

Each format object refers to a paper-type object. Thus, it is actually this pair of objects that specifies how the pages of a document are formatted. The user typically specifies the format options, which translate into format object properties and specifies paper-type options, which translate into paper-type object properties, in the Page Setup Dialog box.

These three objects—the job, format, and paper-type—can refer to other objects, some of which are collections of additional specifications. These other objects and specifications are not required, however, to implement the core printing features.

The references themselves are properties of the job, format, or paper-type objects. The references are mentioned in the following section, which describes each object's properties. The other objects themselves, however, are described as they are used in the chapters "Page Formatting and Dialog Box Customization" and "Advanced Printing Features" in this book.



## Core Printing Features

In addition to manipulating job, format, and paper-type objects, your application must also initialize the printing environment, handle printing-related errors, and handle two situations that can arise when the user invokes a print dialog box:

- n Your application must let QuickDraw GX know which Edit menu items are to be enabled when a QuickDraw GX dialog box is active. Although QuickDraw GX implements the Cut, Copy, Paste, and Clear menu items for you, you must specify which of these items are enabled. If other menu items are enabled, such as Undo, you must also handle the item as well as enable it.
- n Your application must respond to printing event messages, which allows updating the screen when the user moves a dialog box. Printing event messages and movable dialog boxes are described in the chapter “Introduction to Printing With QuickDraw GX” in this book.

Your application should also handle printing from the Finder, which occurs when the user chooses Print from the Finder’s File menu or drags a document onto a desktop printer icon. Finally, your application can also handle printing of existing documents designed for printing with the Macintosh Printing Manager.

The following section, “Core Print Objects,” describes the QuickDraw GX objects needed to implement core printing features. The section “Using Core Printing Features” beginning on page 2-10 provides examples of code that implements these core features.

## Core Print Objects

---

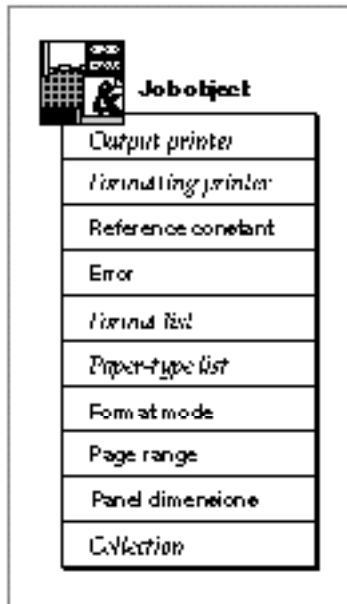
QuickDraw GX printing information is contained in a set of print objects that you associate with a printable document. When you create a job object, QuickDraw GX sets up references to a format object and paper-type object. The initial values for the properties in each of these objects is defined by the printer driver for the default output printer.

The following sections describe specific properties of the job object, the format object, and the paper-type object.

### Job Object Properties

---

A job object has ten accessible properties, as shown in Figure 2-2. Note that, because the data structure of a job object is private, the order of the properties as shown in Figure 2-2 is completely arbitrary. Properties in *italics* indicate references to other objects.

**Figure 2-2** The job object

The properties of a job object are as follows:

- n **Output printer.** A reference to the output printer to which documents are sent for printing. A user specifies an output printer in the Print dialog box. The initial value contained in this property is the default output printer, which is the printer to which documents are sent if the user does not select a different printer.
- n **Formatting printer.** A reference to the printer for which documents are formatted. A user specifies a formatting printer in the Page Setup dialog box. The initial value contained in this property is the default formatting printer, which is used to format documents if the user does not specify a different printer.
- n **Reference constant.** This property contains a reference constant for your application's use. In the reference constant you can associate your own data with a particular job object. For example, you may wish to store a pointer to the document data. Specifying a reference constant for a job object is discussed in the chapter "Advanced Printing Features" in this book.
- n **Error.** This property specifies the most recent error encountered for a particular job object. QuickDraw GX associates printing-related errors with individual job objects. It is necessary for you to check for errors after calling certain functions. Job object errors are discussed in "Error Handling" beginning on page 2-14.
- n **Format mode.** This property specifies the mode associated with a particular job object. QuickDraw GX supports text, PostScript, and graphics direct modes. By default, your application uses the graphics direct mode to print text and graphics. Direct modes allow your application to take advantage of a printer's built-in features, such as fonts and text-streaming capabilities, to provide faster output for users. Using direct mode, however, does not take full advantage of QuickDraw GX features; therefore, the appearance of the document may change when printed in direct mode. The user can

## Core Printing Features

specify a direct mode in the Print dialog box. Direct modes are discussed in the chapter “Advanced Printing Features” in this book.

- n **Format list.** A list of references to format objects. The first reference is to format object that represents the default format. The default format is defined by the printer driver of the default output printer. The user can change the value of the default format in the Page Setup dialog box. Using multiple format objects in a document is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book. Format object properties are discussed in the next section.
- n **Paper-type list.** A list of references to the paper-type objects that are associated with the job's format objects. The user can change the default paper type in the Page Setup dialog box. Using different paper-type objects in a document is discussed in the chapter “Advanced Printing Features” in this book.
- n **Page range.** This property contains the user-specified page range. A user specifies a page range in the Print dialog box. How you determine the page range is discussed in “Printing Documents Using QuickDraw GX” beginning on page 2-20.
- n **Panel dimensions.** This property defines the dimensions of QuickDraw GX dialog box panels. You use this information when you want to locate the position of the cursor within a panel. Panel dimensions are discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.
- n **Collection.** A reference to a job collection object, which stores additional information about the print job. The job collection object is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.

## Format Object Properties

---

A format object contains six accessible properties, as shown in Figure 2-3. Note that, because the data structure of a format object is private, the order of the properties as shown in Figure 2-3 is completely arbitrary. Properties in italics indicate references to other objects.

**Figure 2-3** The format object



The properties of a format object are as follows:

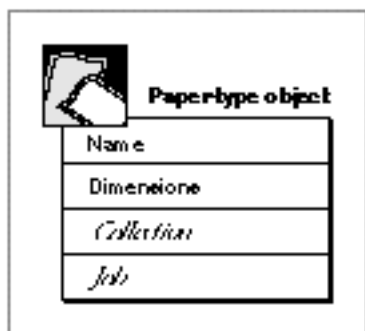
- n **Dimensions.** This property defines the physical dimensions of the paper (the paper size) and the printable area within these dimensions (the page size) after scaling and orientation have been applied. Scaling is the percentage that objects are shrunk or grown when printed. The orientation is either portrait or landscape.
- n **Mapping.** This property defines the mathematical representation of the format object's settings, such as scaling. The mapping property of a format object is discussed in the chapter "Page Formatting and Dialog Box Customization" in this book.
- n **Form.** This property defines a backdrop that can be applied to a set of pages. A **form** is made up of two shape objects—a shape that defines the form and another shape that defines a mask, which represents the erasable area within the form. Forms are discussed in the chapter "Page Formatting and Dialog Box Customization" in this book.
- n **Paper type.** A reference to a paper-type object associated with this format object. Paper-type object properties are discussed in the next section.
- n **Collection.** A reference to a format collection. Through this reference, you can access additional information related to the format collection. This information includes data such as the user-specified orientation (either portrait, landscape, or rotated landscape) from the Page Setup dialog box. The format collection is discussed in the chapter "Page Formatting and Dialog Box Customization" in this book.
- n **Job.** A reference to a job object. Through this reference, you can access the job object associated with a particular format object.

## Paper-Type Object Properties

---

A paper-type object contains four accessible properties, as shown in Figure 2-4. Note that, because the data structure of a paper-type object is private, the order of the properties as shown in Figure 2-4 is completely arbitrary. Properties in *italics* indicate references to other objects.

**Figure 2-4** The paper-type object



The properties of a paper-type object are as follows:

- n **Name.** This property contains the name of a paper type, such as US Letter. A user specifies a paper-type name in the Page Setup or Custom Page Setup dialog box. Paper-type object names are discussed in the chapter “Advanced Printing Features” in this book.
- n **Dimensions.** This property defines the physical dimensions of the paper (the paper size) and the printable area within these dimensions (the page size) before scaling and orientation have been applied. Paper-type object dimensions are discussed in the chapter “Advanced Printing Features” in this book.
- n **Collection.** A reference to a paper-type collection. Through this reference, you can access additional information related to the paper-type object. This information includes such data as paper-type units. The paper-type collection is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.
- n **Job.** A reference to a job object. Through this reference, you can access the job object associated with a particular paper-type object.

## Edit Menu Structure

---

QuickDraw GX supports basic editing commands when a print dialog box is active. The user can Cut, Copy, Paste, and Clear edit text. To handle this task, QuickDraw GX must know the ID of the Edit menu, and the location within the edit menu of the items that correspond to Cut, Copy, Paste, and Clear.

Your application specifies this information in an Edit menu structure, named `gxEditMenuRecord`:

```
struct gxEditMenuRecord{
    short    editMenuID;
    short    cutItem;
    short    copyItem;
    short    pasteItem;
    short    clearItem;
    short    undoItem;
} ;
```

The `editMenuID` field specifies the ID of the Edit menu. The other fields identify the location of items in the Edit menu. For an example of how to set up an Edit menu structure, see “Displaying QuickDraw GX Print Dialog Boxes” beginning on page 2-35.

### Note

QuickDraw GX does not support the Undo item. u

Because QuickDraw GX handles all menu items while a print dialog box is displayed, your application should disable all of its menus, except the Edit menu. It should also disable the About box under the Apple menu. Adjusting menus for movable modal dialog boxes such as print dialog boxes is described in *Inside Macintosh: Macintosh Toolbox Essentials*.

## Using Core Printing Features

---

This section shows how to implement the core printing features in your application. First, you must determine if QuickDraw GX is installed and, if so, set up its environment. Next, when the user creates a document, your application needs to create a job object for the document and maintain other information about the document. The sample code throughout this book uses a structure, `MyDocumentRec`, to keep the needed information in one place:

```
typedef struct MyDocumentRec {
    gxJob      documentJob;      /* the job object bound to the
                                document */
    long       numPages;         /* the number of pages in the
                                document */
    long       curPage;          /* the current page */
    FSSpec     documentFSSpec;   /* the file system specification
                                for the document */
    Str31      documentTitle     /* the title of the document
                                (such as "Untitled") */
    WindowPtr  documentWindow;   /* the window for the document */
    gxViewPort documentViewPort; /* the view port used for
                                drawing within the document
                                window */
    gxShape     documentPage[kMaxPages]; /* the shape data for each
                                page */
    gxFormat    pageFormat[kMaxPages]; /* the format object for each
                                page, if nil use the default
                                format */
} MyDocumentRec, *MyDocumentPtr;
```

This structure is set up to handle one shape per page. Each page may have its own format, although in this chapter only one format is used. The individual fields in the structure are described as they are used in the following sections.

Your application could define a similar structure, or you could maintain the needed information in variables of your choosing. The variable used in this book is `myDocument`, which is defined as follows:

```
MyDocumentRec  myDocument;
```

## Core Printing Features

The following sections show how to

- n initialize QuickDraw GX printing
- n create a job object and initialize the `myDocument` variable
- n handle errors
- n print a document
- n save a job object by flattening it
- n retrieve a job object by unflattening it
- n dispose of a job object and the objects it references
- n obtain format information
- n support print dialog boxes
- n perform printing from the Finder
- n update job object information after resume events
- n print existing documents designed for printing with the Macintosh Printing Manager

## Initializing QuickDraw GX Printing

---

For your application to use QuickDraw GX, the user must be running system software version 7.1 or later. To test for the existence of QuickDraw GX printing features, use the `Gestalt` function. The Gestalt selector is `gestaltPrintingMgrVersion ('pmgr')`. The `Gestalt` function is discussed in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

**Note**

The Gestalt selector for the entire QuickDraw GX feature set is `gestaltGXVersion`. This selector is discussed in *Inside Macintosh: QuickDraw GX Environment and Utilities*. u

After you call the `GXEnterGraphics` function to initialize QuickDraw GX, you call the `GXInitPrinting` function to initialize QuickDraw GX printing features. The `GXEnterGraphics` function is discussed in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Core Printing Features

To terminate printing with QuickDraw GX, you must call the `GXExitPrinting` function. You can only use this function after you have successfully called the `GXInitPrinting` function and before you call the `GXExitGraphics` function to shut down QuickDraw GX:

```
OSErr err;
...
GXEnterGraphics();
err = GXInitPrinting(); /* Set up print facility */
if (!err)
{
    /* The event loop and more initialization goes here */
    ...
}
GXExitPrinting();          /* Close QuickDraw GX printing. */
GXExitGraphics();
```

## Creating a Job Object for a Printable Document

---

For each printable document that a user creates, your application needs to create a corresponding job object. Generally, you should manage job objects on a one-to-one basis with documents. An introduction to manipulating the job object in response to user actions is discussed in the chapter “Introduction to Printing With QuickDraw GX” in this book. Properties of the job object are described in “Job Object Properties” on page 2-5.

Listing 2-1 shows the `MyNewDocument1` function that creates a job object for a printable document and initializes a `MyDocumentRec` structure. The `docName` parameter of the `MyNewDocument1` function is a Pascal string containing the name of the document, and the `myDocument` parameter is a pointer to a `MyDocumentRec` structure. In this example, the document is simplified to handle a maximum of 20 pages.

---

**Listing 2-1**      Creating a job object for a printable document

```
#define kMaxPages    20

OSErr MyNewDocument1(Str31 docName, MyDocumentPtr myDocument)
{
    OSErr    err;
    Rect      bounds;

    myDocument->numPages = 0;          /* there are no pages yet */
    myDocument->curPage = 0;
```



## Core Printing Features

```

/* Create a new job */
err = GXNewJob(&myDocument->documentJob);

if (err == noErr)
{
    /*
       Install your application override for the
       gxPrintingEvent message to display QuickDraw GX movable
       modal dialog boxes.
    */
    GXInstallApplicationOverride(myDocument->documentJob,
                                gxPrintingEvent,
                                MyPrintingEventOverride);

    /*
       Store the document's name. Limit is 31 characters (plus
       a length byte).
    */
    if (docName[0] > 31)
        docName[0] = 31;
    BlockMove(&docName[0], &myDocument->documentTitle[0],
              (long) docName[0] + 1);

    /*
       Additional application-specific document initialization
       can go here, such as the following:
       Create a window and a view port for the document. Store
       the pointer to the MyDocumentRec structure in the
       window's refCon field.
    */
    SetRect(&bounds, 30, 60, 300, 400);
    myDocument->documentWindow = NewCWindow(nil, &bounds,
                                             docName, false, noGrowDocProc, (WindowPtr) -1,
                                             true, (long) myDocument);
    err = MemError();
    if (err == noErr)
    {
        SetPort(myDocument->documentWindow);
        myDocument->documentViewPort =
            GXNewWindowViewPort(myDocument->documentWindow);
    }
}

```

## Core Printing Features

```

        err = GXGetGraphicsError(nil);

        if (err != noErr)
            DisposeWindow(myDocument->documentWindow);
    }
    if (err != noErr) GXDisposeJob(myDocument->documentJob);
}
return err;
}

```

The `MyNewDocument1` function sets the number of pages in the document and the current page number. Note that pages begin at 1 (not from 0 as in an array). The initial value of 0 indicates that there are none.

The `GXNewJob` function creates a job object for the document. If an error does not occur, the `MyNewDocument1` function performs the following tasks:

- n Calls the `GXInstallApplicationOverride` function to install a function that overrides the `gxPrintingEvent` message. This override is needed to handle movable print dialog boxes. The `GXInstallApplicationOverride` function and the `gxPrintingEvent` message are discussed in “Supporting QuickDraw GX Print Dialog Boxes” beginning on page 2-17.
- n Stores the document’s name by calling the `BlockMove` function. The name is passed into the `MyNewDocument1` function.
- n Creates the document’s window by calling the `NewWindow` function and makes it the focus by calling the `SetPort` function.
- n Creates a view port for the window by calling the `GXNewWindowViewPort` function. This view port is used to draw individual shapes on a page and is discussed in the section “Printing Pages by Capturing Shapes” beginning on page 2-22. For information about the `GXNewWindowViewPort` function, see the environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

In the event of an error, the job and window are disposed of, if necessary.

## Error Handling

---

QuickDraw GX provides you with an error-handling method to poll for printing-related errors. In previous versions of the Macintosh printing architecture, errors were handled using the `PrError` function. This function returned the error status. Printing errors were global to an application. In QuickDraw GX, an error is local to a job object.

You can poll for errors in two different ways: immediately after you call a function or after you call groups of functions. QuickDraw GX provides the `GXGetJobError` function to allow you to poll for errors in both ways.

When an error occurs, the error is stored in the error property of the job object. The error is not cleared until you call the `GXGetJobError` function. Thus, `GXGetJobError` returns the first error since the last call to the `GXGetJobError` function.

**IMPORTANT**

If an error condition exists for a job object, QuickDraw GX will not execute other functions associated with the job object until the error condition is cleared. s

You should note that it is necessary for you to check for errors after certain functions. For example, you should always check for errors after calling functions that begin with the word `Start` or functions related to collection objects. Functions related to collection objects are discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.

Polling for errors is a standard Macintosh method used by the Resource Manager and the Macintosh Printing Manager. For information on the Resource Manager, see *Inside Macintosh: More Macintosh Toolbox*. For information on QuickDraw, see *Inside Macintosh: Imaging*.

Listing 2-2 shows an example of polling for errors by calling `GXGetJobError` after individual functions. The error conditions being checked for in the example can arise while executing the print loop. For a discussion of the print loop, see “Printing Documents Using QuickDraw GX” beginning on page 2-20.

---

**Listing 2-2**      Polling for errors after individual functions

```
GXGetJobPageRange(myDocument->documentJob, &firstPage, &lastPage);
err = GXGetJobError(myDocument->documentJob);
if (err == noErr)
{
    if (lastPage > myDocument->numPages)
        lastPage = myDocument->numPages;
    numPages = lastPage - firstPage + 1;

    GXStartJob(myDocument->documentJob,
               myDocument->documentTitle, numPages);
    err = GXGetJobError(myDocument->documentJob);
    if (err == noErr)
    {
        for (pg = firstPage; (err == noErr) && (pg <=
                                     lastPage); pg++)
        {
            GXPrintPage(myDocument->documentJob, pg,
                        GXGetJobFormat(myDocument->documentJob, 1),
                        myDocument->documentPage[pg - 1]);
            err = GXGetJobError(myDocument->documentJob);
        }
    }
}
```

## Core Printing Features

```

        GXFinishJob(myDocument->documentJob);
        err = GXGetJobError(myDocument->documentJob);
    }
}

```

Listing 2-3 shows an example of polling for errors after groups of functions. This example shows how to obtain the dimensions of the paper and page associated with a format object, which is explained on page 2-33. If the `GXGetJobFormat` function returns an error, the `GXGetFormatDimensions` function returns immediately without executing.

---

**Listing 2-3**      Polling for errors after groups of functions

```

OSErr MyGetFormatDimensions(MyDocumentPtr myDocument,
                           gxRectangle *pageBounds,
                           gxRectangle *paperBounds)
{
    long          curPage;
    gxFormat      pgFormat;

    /*
     * Get the format object for the current page. If it is nil, use
     * the default format.
     */
    curPage = myDocument->curPage;
    pgFormat = myDocument->pageFormat[curPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /* Get the bounds of the format object. */
    GXGetFormatDimensions(pgFormat, pageBounds, paperBounds);

    return GXGetJobError(myDocument->documentJob);
}

```

Unless otherwise indicated, errors are generally checked after groups of functions throughout the code samples in this book.

In addition, QuickDraw GX allows you to store an error with a particular job object using the `GXSetJobError` function. This function is useful when you want to abort or cancel spooling, which is how data is sent to the printer driver. Spooling is discussed in the chapter “Introduction to Printing With QuickDraw GX” in this book.

The following statement sets the error condition associated with the job object to the contents of `err`:

```
GXSetJobError(myDocument->documentJob, err);
```

When the error status is tested using `GXGetJobError`, it will return the status set by the `GXSetJobError` function, assuming that another error did not occur between the time the value was set and then retrieved.

## Supporting QuickDraw GX Print Dialog Boxes

---

Dialog boxes for QuickDraw GX printing features are movable modal. A movable modal dialog box is a modal dialog box that contains a title bar by which users can drag the dialog box. This type of dialog box allows users to view windows that would otherwise be obscured by the dialog box. Movable modal dialog boxes are described in *Inside Macintosh: Macintosh Toolbox Essentials*.

To support QuickDraw GX print dialog boxes, your application needs to identify the Edit menu and its menu items, adjust the menu bar to enable or disable appropriate menu items, and respond to the `gxPrintingEvent` message that QuickDraw GX sends to your application.

You make menu adjustments just before you display the dialog box. Examples of setting up the menu bar are shown in the sections “Displaying the Page Setup Dialog Box” beginning on page 2-35 and “Displaying the Print Dialog Box” beginning on page 2-37.

This section shows how to set up the override for the `gxPrintingEvent` message. QuickDraw GX sends this message to your application each time it receives an event, such as a mouse click or a keystroke. Because you want the application to respond to update events so that the window can be redrawn, you must install the application as a handler for the `gxPrintingEvent` message.

You create a function that has the same prototype (the same format of parameters and return value) as the `GXPrintingEvent` function and install it in the message chain. To override the `gxPrintingEvent` message, you specify a pointer to an override function in the `GXInstallApplicationOverride` function. Because dialog boxes are associated with individual job objects, you must call `GXInstallApplicationOverride` after you create each job object.

## Core Printing Features

The override persists until you dispose of the job object or install another override for the `gxPrintingEvent` message. Listing 2-1 on page 2-12 shows the following call in the context of creating a new job object:

```
GXInstallApplicationOverride(myDocument->documentJob,
                             gxPrintingEvent,
                             MyPrintingEventOverride);
```

The `GXInstallApplicationOverride` function has three parameters:

- n A reference to the job object. In Listing 2-1 on page 2-12, it is the job object that was stored when the document was created.
- n The ID of the message to override. In this case, it is `gxPrintingEvent`.
- n The function that responds to the message. In this case, it is `MyPrintingEventOverride`.

The parameters to the override function named `MyPrintingEventOverride` must match those of the `GXPrintingEvent` message override function, which has the following declaration:

```
OSErr GXPrintingEvent (EventRecord *anEventRecord,
                      Boolean filterEvent);
```

The `anEventRecord` parameter is a pointer to the event record, which contains information about what type of event occurred while the print dialog box was being displayed; for example, a mouse click or key-down. The event record also contains additional information associated with the event, such as which key was pressed for a key-down event.

The `filterEvent` parameter specifies whether the event can be filtered. QuickDraw GX sends two `gxPrintingEvent` messages for each event. The first event can be filtered, for example, by calling the `DialogSelect` function to filter non-update events.

**Note**

The Window Manager generates update events to control the appearance of windows on the screen. The `EventRecord` data type, the Window Manager, the `DialogSelect` function, and update events are discussed in *Inside Macintosh: Macintosh Toolbox Essentials*. u

Listing 2-4 shows an override function for the `gxPrintingEvent` message.

---

**Listing 2-4**      Override function for the `gxPrintingEvent` message

```
OSErr MyPrintingEventOverride(EventRecord *anEvent,
                              Boolean filterEvent)
{
    OSErr    err = noErr;

    /* Handle events in whatever way is appropriate. MyDoEvent
       is a generic event handler. Don't pass it events that
       it shouldn't handle while print dialogs are displayed.
    */
    if (!filterEvent)
        switch (anEvent->what)
        {
            case mouseDown:
            case keyDown:
            case autoKey:
                break;

            default:
                err = MyDoEvent(anEvent);
        }
    return err;
}
```

**Note**

You do not need to forward the `gxPrintingEvent` message. u

In Listing 2-4, if the event is not a filter event, `MyDoEvent` is called because the event is probably an update event. The `MyDoEvent` function is the general-purpose, application-specific function that handles all events and is typically called after each `WaitNextEvent`. The `MyDoEvent` function is called from this override to dispatch redrawing of the document's window in response to an update event.

## Printing Documents Using QuickDraw GX

---

There are two approaches you can take to printing a document depending on how you store data. You can either print each page as a single picture shape or print each page by allowing QuickDraw GX to capture multiple shapes. In the later case, you specify when to start and stop capturing shapes that appear on the page.

If your application stores each page as a single picture shape, you should use the `GXPrintPage` function to print each page in a document. In the `GXPrintPage` function, you need to provide QuickDraw GX with the picture shape for each page. A picture shape is a container for other shapes—including other picture shapes, allowing you to create hierarchies of shapes. Picture shapes are discussed in *Inside Macintosh: QuickDraw GX Graphics*.

You may also choose to use the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions to draw and print data. You should use these functions if your application does not store each page as a single picture shape. QuickDraw GX allows you to print in a way similar to how you draw to the screen except that QuickDraw GX captures shapes to send to a print file, such as a spool file or a portable digital document, instead of to a monitor. The `GXDrawShape` function is described in *Inside Macintosh: QuickDraw GX Objects*.

### IMPORTANT

Some QuickDraw GX functions begin with the word `Start` or `Finish`. You must call the corresponding “finish” call only if the “start” call succeeds. For example, after you call the `GXStartPage` function, you should immediately check for errors. You should call the `GXFinishPage` function only if `GXStartPage` did not return an error. s

Regardless of whether you print pages as single picture shapes or print pages by capturing shapes, the basic flow of control is as follows:

- n After the user requests printing, you call the `GXGetJobPageRange` function to obtain the user-specified page range.
- n You use the `GXStartJob` function to begin printing a document with parameters that specify the job object and the name of the user's document. You may also specify the total number of pages the user chose to print or pass 0 if the page count is unknown. In response to the `GXStartJob` call, QuickDraw GX displays the Status dialog box, which contains the current page number and the total page count, if it is not 0.
- n After you finish printing, by either method, you call the `GXFinishJob` function to tell QuickDraw GX that the document is ready to be queued for printing in the background. Note that you should only call the `GXFinishJob` function if the `GXStartJob` function doesn't return an error.



## Printing Pages as Single Picture Shapes

---

This section describes how to use the `GXPrintPage` function to print a user's document. To use this function, you specify the page to print in the `pageNumber` parameter. QuickDraw GX compares the specified page number with the page range chosen by the user and spools the page if it is within the page range. If it is not within range, the page is ignored.

You should loop through each page of a document, calling the `GXPrintPage` function for each page's picture shape. You should check for errors after you print each page and exit the loop if an error arises.

Listing 2-5 gives an example of how to use the `GXPrintPage` function to print a document. In the example, only the default format is used to format each page. To obtain this format, you call the `GXGetJobFormat` function with an index of 1.

---

**Listing 2-5** Using the `GXPrintPage` function to print a document

```
OSErr MyPrintDocument2(MyDocumentPtr myDocument)
{
    OSErr err;
    long firstPage, lastPage, numPages, pg;

    /* Determine which pages the user selected to print. */
    GXGetJobPageRange(myDocument->documentJob,
                      &firstPage,&lastPage);

    if (lastPage > myDocument->numPages)
        lastPage = myDocument->numPages;

    /*
       Calculate the total number of pages to print. If there are
       no errors, begin printing.
    */
    numPages = lastPage - firstPage + 1;
    err = GXGetJobError(myDocument->documentJob);
    if (err == noErr)
    {
        GXStartJob(myDocument->documentJob,
                   myDocument->documentTitle, numPages);
        err = GXGetJobError(myDocument->documentJob);
    }
}
```

## Core Printing Features

```

/*
    Loop through each page. Call the GXPrintPage function for
    each page's picture shape. In this example, we use the
    job's default format to print each page.
*/
if (err == noErr)
{
    for (pg = firstPage; (err == noErr) && (pg <= lastPage);
        pg++)
    {
        GXPrintPage(myDocument->documentJob, pg,
                    GXGetJobFormat(myDocument->documentJob, 1),
                    myDocument->documentPage[pg - 1]);
        err = GXGetJobError(myDocument->documentJob);
    }

    /* Finish printing. */
    if (err == noErr)
    {
        GXFinishJob(myDocument->documentJob);
        err = GXGetJobError(myDocument->documentJob);
    }
}
return err;
}

```

## Printing Pages by Capturing Shapes

---

This section describes how to use the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions to print pages in your application's documents. You use the `GXStartPage` function to tell QuickDraw GX to capture shapes that you draw using the `GXDrawShape` function. You call `GXFinishPage` when you are finished creating the page of output.

In the `GXStartPage` function, you set the page to print in the `pageNumber` parameter. QuickDraw GX compares the specified page number with the page range chosen by the user and spools the page if it is within the page range. If it is not within range, the page is ignored.

## Core Printing Features

In the `GXStartPage` function, you also specify a `viewPortList` parameter, which is the list of view ports to use to capture shapes. The part of the shape that can be drawn through the view port is spooled. In the `numViewPorts` parameter, you specify the number of view ports to use (as specified in the `viewPortList` parameter). QuickDraw GX drawing functions and view port objects are described in *Inside Macintosh: QuickDraw GX Objects*.

**Note**

QuickDraw GX does not use the information in a view port, such as its mapping or clipping properties. It uses a view port only to capture the shape information, such as the geometry and color, as shapes are drawn. For example, you can print as you draw by specifying view ports in the onscreen view group in the call to `GXStartPage`, or you can draw to offscreen view ports to capture shapes without displaying them. In either case, only the information about the shape is spooled. u

Listing 2-6 gives an example of how to print a document using the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions.

---

**Listing 2-6** Using the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions to print a document

```
OSErr MyPrintDocument2(MyDocumentPtr myDocument)
{
    OSErr    err;
    long     firstPage, lastPage, numPages, pg;

    /* Determine which pages the user selected to print. */
    GXGetJobPageRange(myDocument->documentJob, &firstPage,
                     &lastPage);
    if (lastPage > myDocument->numPages)
        lastPage = myDocument->numPages;

    /* Calculate the total number of pages to print.*/
    numPages = lastPage - firstPage + 1;
    err = GXGetJobError(myDocument->documentJob);

    /* Begin printing if there are no errors. */
    if (err == noErr)
    {
        GXStartJob(myDocument->documentJob,
                   myDocument->documentTitle, numPages);
```

## Core Printing Features

```

/*
    For each page, call the GXStartPage function, draw the
    page, and then call the GXFinishPage function. In this
    example, the default format and the document's view
    port are used, drawing only a single shape on each page.
*/
for (pg = firstPage; (err == noErr) && (pg <= lastPage);
    pg++)
{

    /* Start the page. */
    GXStartPage(myDocument->documentJob, pg,
                GXGetJobFormat(myDocument->documentJob, 1),
                1, &myDocument->documentViewPort);
    err = GXGetJobError(myDocument->documentJob);

    /* If there are no errors, draw the data for the page. */
    if (err == noErr)
    {
        GXDrawShape(myDocument->documentPage[pg - 1]);
        err = (OSErr) GXGetGraphicsError(nil);
    }
    if (err == noErr)
        GXFinishPage(myDocument->documentJob);
}

/* Finish printing. */
GXFinishJob(myDocument->documentJob);
err = GXGetJobError(myDocument->documentJob);
}
return err;
}

```

## Saving a Job Object With a Document File

---

There are two approaches you can take to saving a job object with its corresponding document. Either you can create a handle in which to store the job object and then flatten the job object into this handle, or you can specify a pointer to a flattening function to flatten the job object and save its data to disk.

When the user chooses the Save or Save As menu command from the File menu, you should save the document and its corresponding job object to disk. To save a job object, you flatten it. To retrieve a job object, you unflatten it. For an introduction to flattening and unflattening QuickDraw GX print objects, see the chapter “Introduction to Printing With QuickDraw GX” in this book.

When a user saves a document, you may prefer to save a job object in a single handle using the `GXFlattenJobToHdl` function. You may also choose to use the `GXFlattenJob` function to save a job object. You specify a pointer to a flattening function in this function because it requires less memory to save portions of job object data to disk than it does to save the data in a single handle.

### Saving a Job Object in a Single Handle

---

This section describes how to use the `GXFlattenJobToHdl` function to save a job object and its related data.

You should create a handle in which to store the job object and then flatten the job object into this handle. You specify a handle in the `aHandle` parameter to the `GXFlattenJobToHdl` function. QuickDraw GX grows or shrinks the size of the handle you provide to accommodate the size of the job object. You then save the contents of the handle, typically in the document's resource fork.

Listing 2-7 shows how to save a job object in a document using the `GXFlattenJobToHdl` function.

---

**Listing 2-7** Using the `GXFlattenJobToHdl` function to save a job object

```
OSErr MySaveDocument(MyDocumentPtr myDocument)
{
    OSERR      err;
    Handle      theJobData, oldJobData;
    short       dataRefNum = -1;
    short       oldResFile, resRefNum = -1;
    FSSpec      *docFSSpec;

    /*
     * Create a handle in which to store the job object and then
     * flatten the job object into this handle.
     */
    oldResFile = CurResFile();
    theJobData = NewHandle(0);
    err = MemError();
    if (err == noErr)
    {
        GXFlattenJobToHdl(myDocument->documentJob, theJobData);
        err = GXGetJobError(myDocument->documentJob);

        if (err == noErr)
        {
```

## Core Printing Features

```

/* Open the file's data fork and resource fork. */
docFSSpec = &myDocument->documentFSSpec;
err = FSpOpenDF(docFSSpec, fsRdWrPerm, &dataRefNum);
if (err == noErr)
{
    resRefNum = HOpenResFile(docFSSpec->vRefNum,
                             docFSSpec->parID,
                             docFSSpec->name, fsRdWrPerm);
    err = ResError();
}

/* Delete any existing job object resources. */
if (err == noErr)
{
    UseResFile(resRefNum);
    oldJobData = Get1Resource(kMyJobType, kMyJobID);
    if (oldJobData != nil)
    {
        RmveResource(oldJobData);
        UpdateResFile(resRefNum);
        DisposHandle(oldJobData);
    }

    /* Add the new job object resource. */
    AddResource(theJobData, kMyJobType, kMyJobID, "\p");
    err = ResError();
    if (err == noErr)
    {
        WriteResource(theJobData);
        UpdateResFile(resRefNum);
        ReleaseResource(theJobData);
    }
}

/*
    Write the data for a document's pages to the data
    fork. Place your application-specific code here to
    save page data associated with the document.
*/
...
}

```

## Core Printing Features

```

        /* Close the data and resource forks of this document. */
        if (dataRefNum != -1) FSClose(dataRefNum);
        if (resRefNum != -1) CloseResFile(resRefNum);
    }
    else
        DisposHandle(theJobData);
}
UseResFile(oldResFile);
return err;
}

```

### Saving a Job Object Using a Flattening Function

---

This section describes how to use the `GXFlattenJob` function to save a job object. You specify a pointer to a flattening function in the `aPrintingFlattenProc` parameter of this function.

An example of a flattening function named `MyFlattenFunction` that you could write is as follows:

```

OSErr MyFlattenJobFunc(long dataSize, void *data,
                      void *dataRefNum)
{
    long count = dataSize;
    return FWrite((short) dataRefNum, &count, data);
}

```

QuickDraw GX calls your flattening function multiple times as it saves job object data to disk. The `dataSize` parameter specifies the number of bytes for this segment of the job object data. The `data` parameter specifies a pointer to the segments of job object data to write out. The `dataRefNum` parameter specifies the file reference number of the open file to which you want to write.

Listing 2-8 shows how to save a job object using the `GXFlattenJob` function.

---

**Listing 2-8** Using the `GXFlattenJob` function to save a job object

```
OSErr MySaveJobInDataFork(MyDocumentPtr myDocument,
                          short dataRefNum)
{
    OSErr    err;

    /*
     * Reset the file's position to the beginning of the data fork
     * and write the flattened job object there.
     */
    err = SetFPos(dataRefNum, fsFromStart, 0);
    if (err == noErr)
    {
        GXFlattenJob(myDocument->documentJob,
                     (gxPrintingFlattenProc) MyFlattenJobFunc,
                     dataRefNum);
        err = GXGetJobError(myDocument->documentJob);
    }
    return err;
}
```

## Disposing of a Job Object When Closing a Document

---

When the user chooses the Close menu command from the File menu to close a document, you need to dispose of its job object. You should not dispose of a job object while its document is open.

For each page in a document, you should dispose of the page's shape. You can then call the `GXDisposeJob` function to dispose of a document's job object and associated format objects. Listing 2-9 shows how to dispose of a job object when a user closes a document.



**Listing 2-9** Disposing of a job object when you close a document

---

```

OSErr MyCloseDocument(MyDocumentPtr myDocument)
{
    OSERR    err = noErr, jobErr;
    long     pg;

    /* Dispose of each page's shape */
    for (pg = 1; pg <= myDocument->numPages; pg++)
        GXDisposeShape(myDocument->documentPage[pg-1]);

    /* Dispose of the document's corresponding job object. */
    err = GXDisposeJob(myDocument->documentJob);

    /*
       Place any application-specific code here to close a
       document.
    */
    ...
    DisposeWindow(myDocument->documentWindow);
    return err;
}

```

**Note**

The `GXDisposeJob` function returns an error because errors are job-oriented. You cannot query a job object for errors once you have disposed of it. <sup>u</sup>

## Retrieving a Job Object When Opening a Document

---

When the user chooses the Open menu command from the File menu to open a document, you need to retrieve its job object. To retrieve a job object, you unflatten it using one of the QuickDraw GX unflattening functions. For an introduction to flattening and unflattening QuickDraw GX print objects, see the chapter “Introduction to Printing With QuickDraw GX” in this book.

There are two methods to retrieving a job object depending on how you have previously saved it. If you saved the job object using the `GXFlattenJobToHdl` function, you should retrieve it using the `GXUnflattenJobFromHdl` function. If you saved the job object using the `GXFlattenJob` function, you should retrieve it using the `GXUnflattenJob` function. For details on the `GXFlattenJobToHdl` and `GXFlattenJob` functions, see “Saving a Job Object With a Document File,” which begins on page 2-24.

## Retrieving a Job Object From a Handle

---

This section describes how to use the `GXUnflattenJobFromHdl` function to retrieve a job object and its related data.

When a user chooses the Open menu command from the File menu, you should open the document and retrieve its previously saved job object. To do so, you open the document's data fork and resource fork. The `MyOpenDocument` function in Listing 2-10 accomplishes this.

If there are no errors, you should specify the document's file system specification information, its title, and its window's title. If there is a job object resource saved in the resource file, you should load it and unflatten it using the `GXUnflattenJobFromHdl` function.

After the job object is unflattened, you can load the data for the document's pages. Finally, you should close the document's data fork and resource fork. Listing 2-10 shows how to open a document and retrieve its job object using the `GXUnflattenJobFromHdl` function.

---

**Listing 2-10** Using the `GXUnflattenJobFromHdl` function to retrieve a job object

```
OSErr MyOpenDocument(MyDocumentPtr myDocument)
{
    OSErr          err;
    Handle          theJobData;
    short           oldResFile;
    short           dataRefNum = -1, resRefNum = -1;
    StandardFileReply sfReply;
    SFTYPEList      myTypeList;

    /* Let the user select a document to open. */
    oldResFile = CurResFile();

    myTypeList[0] = kMyDocType;
    StandardGetFile(nil, 1, &myTypeList, &sfReply);
    if (!sfReply.sfGood)
        return noErr;

    /* Open the selected file's data fork and resource fork. */
    err = FSpOpenDF(&sfReply.sfFile, fsRdWrPerm, &dataRefNum);
    if (err == noErr)
    {
        resRefNum = HOpenResFile(sfReply.sfFile.vRefNum,
                                sfReply.sfFile.parID,
                                sfReply.sfFile.name, fsRdPerm);
    }
}
```

## Core Printing Features

```

        err = ResError();
    }
    if (err) return err;

    /*
     * If no error, set the document's file system specification
     * information, its title, and its window's title.
     */
    BlockMove(&sfReply.sfFile, &myDocument->documentFSSpec,
              sizeof(FSSpec));
    BlockMove(&sfReply.sfFile.name, myDocument->documentTitle,
              (long) sfReply.sfFile.name[0] + 1);
    SetWTitle(myDocument->documentWindow,
              myDocument->documentTitle);

    /*
     * If there's a job object resource saved,
     * load and unflatten it.
     */
    UseResFile(resRefNum);
    theJobData = Get1Resource(kMyJobType, kMyJobID);
    if (theJobData != nil)
    {
        GXUnflattenJobFromHdl(myDocument->documentJob, theJobData);
        err = GXGetJobError(myDocument->documentJob);
        ReleaseResource(theJobData);
    }

    /*
     * Place your application-specific code here to load
     * other data associated with the document's pages.
     */
    ...

    /* Close the data fork and resource fork of this document. */
    if (dataRefNum != -1) FSClose(dataRefNum);
    if (resRefNum != -1) CloseResFile(resRefNum);
    UseResFile(oldResFile);
    return err;
}

```

## Retrieving a Job Object Using an Unflattening Function

---

This section describes how to use the `GXUnflattenJob` function to retrieve a job object. You specify a pointer to an unflattening function in the `aPrintingFlattenProc` parameter of the `GXUnflattenJob` function.

An example of an unflattening function named `MyUnflattenFunction` that you could write is as follows:

```
OSErr MyUnflattenJobFunc(long dataSize, void *data,
                        void *dataRefNum)
{
    long count = dataSize;
    return FSRead((short) dataRefNum, &count, data);
}
```

QuickDraw GX calls your unflattening function multiple times as it retrieves job object-related data from disk. The `dataSize` parameter specifies the number of bytes for this segment of the job object data. The `data` parameter specifies a pointer to the segments of job object data to read. The `dataRefNum` parameter specifies the file reference number of the open file from which you want to read.

Listing 2-11 shows how to retrieve a job object using the `GXUnflattenJob` function.

---

**Listing 2-11** Using the `GXUnflattenJob` function to retrieve a job object

```
OSErr MyLoadJobFromDataFork(MyDocumentPtr myDocument,
                          short dataRefNum)
{
    OSErr err;

    /*
     * Reset the file's position to the beginning of the data fork,
     * read and then unflatten the job object from there.
     */
    err = SetFPos(dataRefNum, fsFromStart, 0);
    if (err == noErr)
    {
        GXUnflattenJob(myDocument->documentJob,
                      (gxPrintingFlattenProc) MyUnflattenJobFunc,
                      (void *) dataRefNum);
        err = GXGetJobError(myDocument->documentJob);
    }
    return err;
}
```

## Obtaining Object References

---

A job object can reference several format objects. Once you know which format object you want, you can access its properties. QuickDraw GX provides the `GXGetFormatJob` function to determine which job object is associated with a particular format object. Even if you know the format's job, you may still want to examine all references to the job's format objects. You can obtain these references with the `GXGetJobFormat` function.

Listing 2-12 shows an example that uses the `GXGetFormatJob` function to obtain the job object that references a format object and then loops through all the job's format objects using the `GXGetJobFormat` function. The example's function, `MyGetFormatIndex`, returns the format's position, or index value, of the specified format object in the job's list of format objects.

---

**Listing 2-12** Using the `GXGetFormatJob` function to obtain a job object

```
long MyGetFormatIndex(gxFormat myFormat)
{
    gxJob    formatsJob;
    long     idx, numFormats;

    /*
     * Obtain the job object and count of the number of format objects
     * it references.
     */
    formatsJob = GXGetFormatJob(myFormat);
    numFormats = GXCountJobFormats(formatsJob);

    /*
     * Compare each of the references to locate the specified format
     * object and return the current format object index.
     */
    for (idx = 1; idx <= numFormats; ++idx)
        if (myFormat == GXGetJobFormat(formatsJob, idx))
            return idx;
}
```

## Obtaining Information From a Format Object

---

This section provides an example of how to obtain information from a format object. QuickDraw GX provides functions that allow you to get, and in some cases set, the values of printing-related object properties.

This example uses the `GXGetFormatDimensions` function, which returns the dimensions property of a format object. The dimensions property includes the physical dimensions of the paper (the paper size) and the printable area within these

dimensions (the page size) after scaling and orientation have been applied. For a discussion of how the dimensions can be scaled or otherwise changed, see the chapter “Page Formatting and Dialog Box Customization” in this book.

Listing 2-13 shows how to use the `GXGetFormatDimensions` function to obtain a format object’s dimensions property.

---

**Listing 2-13** Using the `GXGetFormatDimensions` function

```
OSErr MyGetFormatDimensions(MyDocumentPtr myDocument,
                            gxRectangle *pageBounds,
                            gxRectangle *paperBounds)
{
    long          curPage;
    gxFormat      pgFormat;

    /*
     * Get the format object for the current page. If it is nil, use
     * the default format.
     */
    curPage = myDocument->curPage;
    pgFormat = myDocument->pageFormat[curPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /* Get the bounds of the format object. */
    GXGetFormatDimensions(pgFormat, pageBounds, paperBounds);

    return GXGetJobError(myDocument->documentJob);
}
```

**Note**

The `GXGetFormatDimensions` function returns both the page size and the paper size of a particular document. Most applications are generally interested in only the page size, so QuickDraw GX allows you to pass `nil` for the pointer to the paper size. u

## Displaying QuickDraw GX Print Dialog Boxes

---

You call functions to display most QuickDraw GX print dialog boxes. You use the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box, and you use the `GXJobPrintDialog` function to display the Print dialog box. You use the `GXFormatDialog` function to display the Custom Page Setup dialog box, which is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.

### Displaying the Page Setup Dialog Box

---

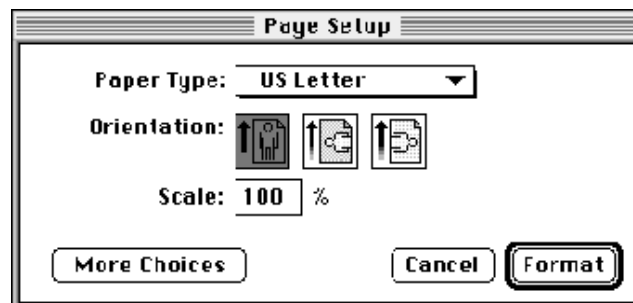
When the user chooses the Page Setup menu command from the File menu, you call the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box. In this dialog box, the user can specify formatting information for the default format. For example, the user can specify the paper type, orientation, and scaling.

QuickDraw GX stores a user’s responses to some dialog items in the Page Setup dialog box in a format collection. QuickDraw GX stores default items, such as these, for you automatically. The format collection is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.

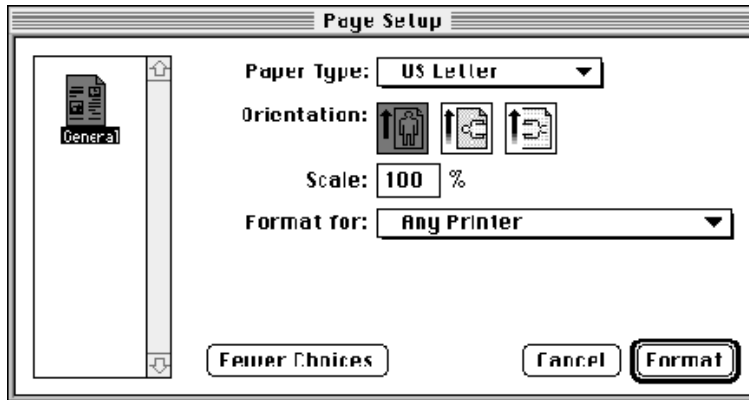
Figure 2-5 shows the Page Setup dialog box the user sees when you call the `GXJobDefaultFormatDialog` function.

---

**Figure 2-5** The Page Setup dialog box



If the user chooses More Choices in the Page Setup dialog box, QuickDraw GX expands the dialog box. Figure 2-6 shows the expanded Page Setup dialog box. The expanded dialog box in this figure only contains one panel, the General panel. A printer driver, printing extension, or application can customize the dialog box to add additional panels. For more information about adding panels, see the chapter “Page Formatting and Dialog Box Customization” in this book.

**Figure 2-6** The expanded Page Setup dialog box

Listing 2-14 shows the `MyFormatDialog` function, which calls the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box. The Edit menu structure, `gxEditMenuRecord`, is set up before the dialog box is displayed. For information about the Edit menu structure, see “Edit Menu Structure” beginning on page 2-9. If the user chooses the Format button and there are no errors, document formatting can proceed.

**Listing 2-14** Displaying the Page Setup dialog box

```
#define mEdit      128

#define kUndo      1
#define kCut       3
#define kCopy      4
#define kPaste     5
#define kClear     6
...
OSErr MyFormatDialog(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxDialogResult  result;
    gxEditMenuRecord editMenuRec;

    /* Fill in the location of your application's Edit menu items. */

    editMenuRec.editMenuID = mEdit;
    editMenuRec.cutItem    = kCut;
    editMenuRec.copyItem   = kCopy;
    editMenuRec.pasteItem   = kPaste;
```



## Core Printing Features

```

editMenuRec.clearItem    = kClear;
editMenuRec.undoItem     = kUndo;

/* Display the Page Setup dialog box. */
result = GXJobDefaultFormatDialog(myDocument->documentJob,
                                   &editMenuRec);
err = GXGetJobError(myDocument->documentJob);

/*
   If the user chooses the Format button and there are no
   errors, perform document formatting.
*/
if ((err == noErr) && (result == gxOKSelected))
{
    /*
       Place your application-specific code here if you need
       to repaginate the document.
    */
    ...
}

return err;
}

```

## Displaying the Print Dialog Box

---

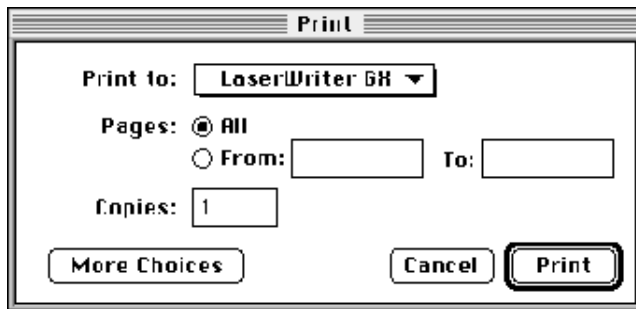
When the user chooses the Print menu command from the File menu, you call the `GXJobPrintDialog` function to display the Print dialog box. In this dialog box, the user can specify information related to actual printing of the document. For example, in the panels of the Print dialog box the user can specify the printer, print quality, number of copies to print, page range, automatic or manual paper feed, and whether a document should be sent to a printer or a file.

QuickDraw GX stores a user's responses to some dialog items in the Print dialog box in a job collection. The job collection is discussed in the chapter "Page Formatting and Dialog Box Customization" in this book.

## Core Printing Features

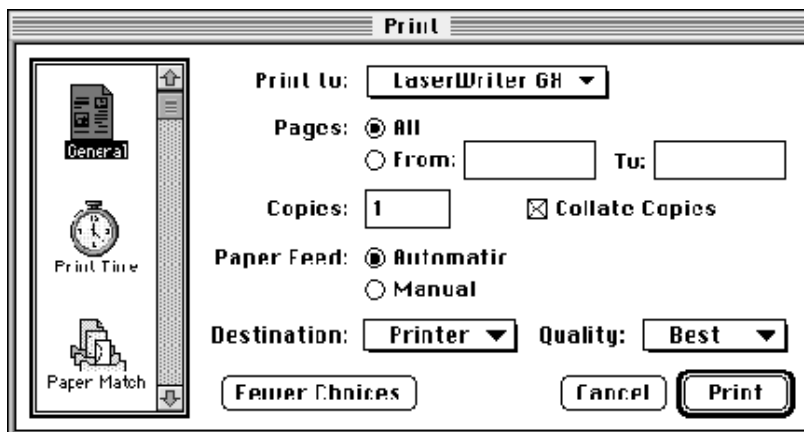
Figure 2-7 shows the Print dialog box the user sees when you call the `GXJobPrintDialog` function.

**Figure 2-7** The Print dialog box



If the user chooses More Choices in the Print dialog box, QuickDraw GX expands the dialog box. Figure 2-8 shows the expanded Print dialog box. The expanded dialog box includes the standard panels (General, Print Time, and Paper Match), and any panels added by the application, printing extensions, or a printer driver.

**Figure 2-8** The expanded Print dialog box



Listing 2-15 shows the `MyPrintDialog` function, which calls the `GXJobPrintDialog` function to display the Print dialog box. The Edit menu structure, `gxEditMenuRecord`, is set up before the dialog box is displayed. For information about the Edit menu structure, see “Edit Menu Structure” beginning on page 2-9. If the user chooses the Print button and there are no errors, printing can proceed.

**Listing 2-15**    Displaying the Print dialog box

---

```

OSErr MyPrintDialog(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxDialogResult  result;
    gxEditMenuRecord editMenuRec;

    /* Fill in the location of your application's Edit menu items. */

    editMenuRec.editMenuID = mEdit;
    editMenuRec.cutItem    = kCut;
    editMenuRec.copyItem   = kCopy;
    editMenuRec.pasteItem  = kPaste;
    editMenuRec.clearItem  = kClear;
    editMenuRec.undoItem   = kUndo;

    /* Display the Print dialog box. */
    result = GXJobPrintDialog(myDocument->documentJob,
                              &editMenuRec);
    err = GXGetJobError(myDocument->documentJob);

    /*
     * If the user chooses the Print button and there are no errors,
     * call your printing function to print the pages.
     */
    if ((err == noErr) && (result == gxOKSelected))
        err = MyPrintDocument(myDocument);

    return err;
}

```

## Supporting Printing From the Finder

---

A user can print from the Finder in two ways. A user can select a document and then choose the Print menu command from the File menu, or the user can drag a document to a desktop printer icon. To support printing from the Finder, your application must respond to the Print Documents ('pdoc') Apple event. Apple events provide your application with a standard mechanism for communicating with other applications.

To handle the Print Documents event, your application should print the documents specified in the Apple event. You can determine whether a document was dragged to a desktop printer icon by checking the `keyOptionalKeywordAttr` attribute of the Print Documents Apple event. Your application extracts this information and then prints the specified documents. Your application should not open any windows for the documents.

The Print Documents Apple event is discussed in the Apple events chapter of *Inside Macintosh: Interapplication Communication*.

Your application is responsible for determining the output printer on which to print the document. When a user drags a document to a desktop printer icon, your application must call the `GXSelectJobOutputPrinter` function to specify the output printer on which to print the selected document. This call is necessary because the document may have been printed previously and that job information may have been saved with the document. The `GXSelectJobOutputPrinter` function allows you to reselect the printer.

Listing 2-16 shows how to respond to the Print Documents Apple event and specify an output printer.

---

**Listing 2-16** Responding to the Print Documents Apple event and specifying an output printer

```
pascal OSErr MyHandlePDOC(AppleEvent *theAppleEvent,
                          AppleEvent *reply, long myRefCon)
{
    OSErr          err;
    AEDescList     docList, dtpList;
    FSSpec         myFSS, dtpFSS;
    long           itemsInList, i;
    AEKeyword      theKeyword;
    DescType       typeCode;
    Boolean        draggedToDTP = false;
    Size           actualSize;
    MyDocumentRec  myDocument;

    /* Get the document list. */
    err = AEGetParamDesc(theAppleEvent, keyDirectObject,
                        typeAEList, &docList);
    if (err) return err;

    /*
     * Check to see if the user dragged the document to a desktop
     * printer.
     */
    err = AEGetParamDesc(theAppleEvent, keyOptionalKeywordAttr,
                        typeAEList, &dtpList);
    if (err == noErr) draggedToDTP = true;
```

## Core Printing Features

```

/*
    Make sure you've accounted for all of the parameters passed
    and count the number of documents specified.
*/
err = MyCheckAEPARAMS(theAppleEvent);
if (err) return err;
err = AECOUNTITEMS(&docList, &itemsInList);
if (err) return err;

/*
    If the user dragged the document to a desktop printer, get the
    name of the desktop printer and throw away its description
    list.
*/
if (draggedToDTP)
{
    err = AEGETHPTR(&dtpList, 1, typeFSS, &theKeyword,
                   &typeCode, (Ptr) &dtpFSS,
                   sizeof(FSSpec), &actualSize);
    AEDISPOSEDESC(&dtpList);
}

/*
    For each entry in the document list, load it, print it, and
    close it.
*/
for (i = 1; i <= itemsInList; err == noErr; i++)
{
    err = AEGETHPTR(&docList, i, typeFSS, &theKeyword,
                   &typeCode, (Ptr) &myFSS, sizeof(FSSpec),
                   &actualSize);

    if (err == noErr)
    {
        /* Load the document. */
        err = MyNewDocument("\p", &myDocument);
        if (err == noErr)
        {
            err = MyFSOpenDocument(&myDocument, &myFSS);
            if (err == noErr)

```

## Core Printing Features

```

        /*
         * If the user dragged the document to a desktop
         * printer, select this printer as the output printer
         * for each job object.
         */
        {
            if (draggedToDTP)
                GXSelectJobOutputPrinter(myDocument.documentJob,
                                         dtpFSS.name);

            err = MyPrintDocument(&myDocument);
        }

        /* Close the document once it's printed. */
        MyCloseDocument(&myDocument);
    }
}

/* When you're done, throw away the document list. */
AEDisposeDesc(&docList);
return err;
}

```

## Updating Job Object Information

---

When you receive a resume event, you should use the `GXUpdateJob` function to update the job object because the printing environment may have changed while the user was switched out of your application. For example, the user may have changed the desktop printer's settings, such as paper-tray information, while using another application.

Listing 2-17 shows an example of how to update the job object for a document. The `GXUpdateJob` function is called from the `MyDoEvent` function in response to a resume event.

**Listing 2-17** Updating a job when receiving resume events

---

```

OSErr MyDoEvent(EventRecord *event)
{
    OSErr          err = noErr;
    WindowPtr      curWindow;
    MyDocumentPtr  windowDoc;

    switch (event->what)
    {
        /*
         * Application-specific code to handle mouse-down events,
         * update events, and so on.
         */
        case osEvt:
            switch ((event->message >> 24) & 0xFF)
            {

                case suspendResumeMessage:
                    SetCursor(&qd.arrow);

            /* On a suspend event, coerce the scrap. */
                if ((event->message & resumeFlag) == 0)
                {
                    ZeroScrap();
                    TEToScrap();
                }
                else
                {

            /*
             * On a resume event, call GXUpdateJob on all of the documents'
             * job objects. The user may have just changed something which
             * affects the job objects, such as the size of the paper in the
             * printer.

```

## Core Printing Features

Since your application stores the document pointers in the reference constant fields of the documents' windows, loop through each window, extract the document pointers, and update the associated job objects.

```

*/
        if (event->message & convertClipboardFlag)
            TEFromScrap();
        curWindow = FrontWindow();
        while (curWindow != nil)
        {
            if (((WindowPeek) curWindow)->windowKind ==
                                                userKind)
            {
                windowDoc = (MyDocumentPtr)
                                GetWRefCon(curWindow);
                GXUpdateJob(windowDoc->documentJob);
            }
            curWindow = (WindowPtr) ((WindowPeek)
                                    curWindow)->nextWindow;
        }
        break;
    }
    break;

    /*
        Application-specific code to handle high-level events.
    */
}
return err;
}

```

## Printing Macintosh Printing Manager Documents

---

Documents printed with applications that use the Macintosh Printing Manager can be printed on a system with QuickDraw GX installed without the application being aware that QuickDraw GX is installed. Printing in this way, however, does not allow the application to take advantage of QuickDraw GX printing features, such as additional options provided in QuickDraw GX print dialog boxes, formatting, customization, and so on.



You can modify an existing application to allow it to print a document designed for printing with the Macintosh Printing Manager by determining whether QuickDraw GX is installed and, if it is, performing these steps:

1. Convert the print record associated with a Macintosh Printing Manager document into a job object.
2. Install the QuickDraw GX Translator to convert the results of QuickDraw functions into special QuickDraw GX shape objects that are used to spool QuickDraw output.
3. Execute your print loop. See the section “Printing Documents Using QuickDraw GX” beginning on page 2-20 for an example.
4. Remove the QuickDraw GX Translator.

To convert the print record associated with a Macintosh Printing Manager document into a job object, use the `GXConvertPrintRecord` function. Listing 2-18 shows how to use the `GXConvertPrintRecord` function.

---

**Listing 2-18**     Converting a print record into a job object

```
OSErr MyPrintRecordToJob(MyDocumentPtr myDocument, THPrint hPrint)
{
    /*
     * Convert the print record and store its settings in
     * the specified job object. Dispose of its handle.
     */
    GXConvertPrintRecord(myDocument->documentJob, hPrint);
    DisposHandle((Handle) hPrint);

    return GXGetJobError(myDocument->documentJob);
}
```

In addition to converting the print record, you must also translate QuickDraw data using the QuickDraw GX Translator. You call the `GXInstallQDTranslator` function to install the translator, and you call the `GXRemoveQDTranslator` function when you are finished with the translation. The QuickDraw GX Translator and these functions are described in the QuickDraw GX environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

After you call `GXInstallQDTranslator`, you proceed to print using the normal print loop; for example, by calling `GXStartPage` to start a new page and `GXFinishPage` to finish it. The results of any QuickDraw function is translated and the output is spooled. When you are finished printing, call `GXRemoveQDTranslator` to end translation.

## Core Printing Features Reference

---

This section describes the data types, constants, and functions that are specific to QuickDraw GX core printing features.

The “Constants and Data Types” section shows the Gestalt selector enumeration for QuickDraw GX printing features, the data types for QuickDraw GX printing-related objects, the Edit menu structure, and the dialog box result enumeration.

The “Functions” section describes functions for initializing and terminating printing features, handling errors, creating and managing job objects, printing using QuickDraw GX, obtaining information on printing-related objects, displaying the Page Setup and Print dialog boxes, and converting a print record into a job object.

The “Application-Defined Functions” section shows sample functions for flattening and unflattening job objects.

### Constants and Data Types

---

This section describes the data types and constants that you use to initialize QuickDraw GX printing features, reference QuickDraw GX printing-related objects, and support QuickDraw GX print dialog boxes.

You can use the Gestalt selector enumeration to test for the existence of QuickDraw GX printing features.

The QuickDraw GX printing-related object structures are private. You can access print objects through references.

You can use the Edit menu structure to specify the location of the Edit menu and its menu items when displaying print dialog boxes.

You can use the dialog box result enumeration to store the user’s response to QuickDraw GX print dialog boxes.

## Gestalt Selectors for Printing

---

To test for the existence of QuickDraw GX printing features, use the `Gestalt` function. The Gestalt selectors for the QuickDraw GX printing manager version and QuickDraw GX are defined as follows:

```
#define gestaltGXPrintingMgrVersion 'pmgr'
#define gestaltGXVersion            'qdgx'
```

The `Gestalt` function is discussed in *Inside Macintosh: Operating System Utilities*.

## QuickDraw GX Printing-Related Objects

---

QuickDraw GX provides you with access to printing-related objects through references. The contents of the structures are private.

You access a job object through a job object reference:

```
typedef struct gxPrivateJobRecord *gxJob;
```

You access printer objects through a printer object reference:

```
typedef struct gxPrivatePrinterRecord *gxPrinter;
```

You access a format object through a format object reference:

```
typedef struct gxPrivateFormatRecord *gxFormat;
```

You access a paper-type object through a paper-type object reference:

```
typedef struct gxPrivatePaperTypeRecord *gxPaperType;
```

You access a print file object through a print file object reference:

```
typedef struct gxPrivatePrintFileRecord *gxPrintFile;
```

QuickDraw GX also provides the job, format, and paper-type collection objects. You access collection objects through a collection object reference:

```
typedef struct PrivateCollectionRecord *Collection;
```

## Edit Menu Location

---

When displaying QuickDraw GX print dialog boxes, your application needs to specify the location of the Edit menu and its menu items. Your application specifies the location of the Edit menu and its menu items in the Edit menu structure. The Edit menu structure is defined as follows:

```
struct gxEditMenuRecord {
    short    editMenuID;
    short    cutItem;
    short    copyItem;
    short    pasteItem;
    short    clearItem;
    short    undoItem;
} gxEditMenuRecord;
```

### Field descriptions

<code>editMenuID</code>	Your application's resource ID for the Edit menu.
<code>cutItem</code>	The position of the cut menu item under the Edit menu.
<code>copyItem</code>	The position of the copy menu item under the Edit menu.
<code>pasteItem</code>	The position of the paste menu item under the Edit menu.
<code>clearItem</code>	The position of the clear menu item under the Edit menu.
<code>undoItem</code>	The position of the undo menu item under the Edit menu.

## Dialog Box Results

---

QuickDraw GX print dialog boxes support dialog box results. Results are defined in the dialog box result enumeration.

```
enum {
    gxCancelSelected  = (gxDialogResult) 0,
    gxOKSelected      = (gxDialogResult) 1,
    gxRevertSelected  = (gxDialogResult) 2
};

typedef long gxDialogResult;
```

## Core Printing Features

**Constant descriptions**`gxCancelSelected`

Represents a cancelation of the dialog box without action being taken, such as when the user chooses Cancel or presses Escape while in a dialog box.

`gxOKSelected`

Represents a confirmation, such as when the user chooses Format in the Page Setup dialog box.

`gxRevertSelected`

Represents a request to undo one or more actions, such as when the user chooses Remove to remove a page format while in the Custom Page Setup dialog box.

## Functions

---

This section describes the functions for initializing and terminating printing features, handling errors, creating and managing job objects, printing using QuickDraw GX, obtaining information on print objects, displaying print dialog boxes, and converting a print record into a job object.

Included with each function description is a list of specific result codes returned by QuickDraw GX. In addition to these result codes, you may also receive file-system, memory, and resource errors. For a complete listing of specific file-system, memory, and resource errors, see *Inside Macintosh: C Summary* or *Inside Macintosh: Pascal Summary*.

You should note that not all possible result codes for a particular function are included in function descriptions within this section. For example, the Message Manager, described in *Inside Macintosh: QuickDraw GX Environment and Utilities*, allows QuickDraw GX functions to send specific messages to your application. These messages can also generate errors.

**IMPORTANT**

All printing functions in QuickDraw GX, with the exception of the `GXGetJobError` function, may move Macintosh memory. The `GXGetJobError` function, however, relies on data that may also move. Therefore, your application should never call a QuickDraw GX printing-related function at interrupt time. *s*

## Initializing and Terminating QuickDraw GX Printing Features

---

After you call the `GXEnterGraphics` function to initialize QuickDraw GX, you can call the `GXInitPrinting` function to initialize printing features within QuickDraw GX.

When you have successfully called the `GXInitPrinting` function and you need to terminate printing features within QuickDraw GX, you must call the `GXExitPrinting` function.

### GXInitPrinting

---

You can use the `GXInitPrinting` function to initialize printing features within QuickDraw GX.

```
OSErr GXInitPrinting (void);
```

*function result* An error code of type `OSErr`.

#### DESCRIPTION

Before you call the `GXInitPrinting` function, you must call the `GXEnterGraphics` function to initialize QuickDraw GX. You should also use the `Gestalt` function to determine whether QuickDraw GX printing features are available on the user's system. The `Gestalt` selector is `'pmgr'`.

#### SPECIAL CONSIDERATIONS

If the `GXInitPrinting` function returns an error, you should not attempt to call other QuickDraw GX printing-related functions.

#### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

#### SEE ALSO

The `GXEnterGraphics` function that initializes QuickDraw GX is described in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To terminate printing features in QuickDraw GX, use the `GXExitPrinting` function, which is described in the next section.

## GXExitPrinting

---

You can use the `GXExitPrinting` function to terminate printing features within QuickDraw GX.

```
OSErr GXExitPrinting (void);
```

*function result* An error code of type `OSErr`.

### DESCRIPTION

The `GXExitPrinting` function terminates printing features within QuickDraw GX only after you have successfully called the `GXInitPrinting` function. You cannot call QuickDraw GX printing functions after you call the `GXExitPrinting` function.

You must call the `GXExitPrinting` function before you call the `GXExitGraphics` function to shut down QuickDraw GX.

Before you call the `GXExitPrinting` function, you should dispose of all QuickDraw GX printing-related objects. If you want to use these objects again, you should save them.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

### SEE ALSO

For information about saving printing-related objects, see “Saving a Job Object With a Document File” beginning on page 2-24.

The `GXExitGraphics` function is described in the environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Handling Errors

---

QuickDraw GX printing features allow you to poll for errors in two ways: immediately after you call a function or after you call groups of functions. QuickDraw GX provides the `GXGetJobError` function to allow you to poll for errors in both ways.

To allow your application to manage separate documents, errors are local to a job object. To store an error with a particular job object, you use the `GXSetJobError` function.

### GXGetJobError

---

You can use the `GXGetJobError` function to obtain the first error encountered for a particular job object since the last call to `GXGetJobError`.

```
OSErr GXGetJobError (gxJob aJob);
```

`aJob`                    A reference to the job object whose most recent error you want to obtain.

*function result*    An error code of type `OSErr`.

#### DESCRIPTION

The `GXGetJobError` function returns printing-related errors associated with a job object. Initially, you can call this function to obtain the current error code. If you immediately call this function a second time, it returns `noErr`.

You can use the `GXSetJobError` function to store an error in a specific job object.

#### SPECIAL CONSIDERATIONS

After an error occurs, calls to QuickDraw GX printing-related functions associated with the specified job object return immediately without executing, until the `GXGetJobError` function is called.

The `GXGetJobError` function does *not* move Macintosh memory; however, your application should not call this function at interrupt time, because it relies on data structures that may move.

#### SEE ALSO

Error-handling methods using the `GXGetJobError` function are described in “Error Handling,” which begins on page 2-14.

The `GXSetJobError` function is described in the next section.



## **GXSetJobError**

---

You can use the `GXSetJobError` function to store an error in the provided job object.

```
void GXSetJobError (gxJob aJob, OSErr anError);
```

<code>aJob</code>	A reference to the job object in which to store the error.
<code>anError</code>	The error to store.

### **DESCRIPTION**

The `GXSetJobError` function stores an error with a particular job object. This function is useful when you want to abort or cancel spooling.

Most applications do not need to use this function because QuickDraw GX sets the error for you. You might want to use it, however, to artificially raise an error condition.

### **SPECIAL CONSIDERATIONS**

An existing error is replaced when you call the `GXSetJobError` function. If you wish to save a previous error, you must call the `GXGetJobError` function to obtain an error prior to calling the `GXSetJobError` function.

### **RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## Creating and Managing Job Objects

---

When a user creates a new document, you need to create a corresponding job object using the `GXNewJob` function. When a user closes a printable document, you need to dispose of its corresponding job object using the `GXDisposeJob` function.

When a user saves a printable document, you need to flatten its job object using either the `GXFlattenJobToHdl` function or the `GXFlattenJob` function. When a user opens a printable document, you need to retrieve its job object using either the `GXUnflattenJobFromHdl` function or the `GXUnflattenJob` function.

When you receive a resume event, you should use the `GXUpdateJob` function to update the job object because the printing environment may have changed.

## GXNewJob

---

You can use the `GXNewJob` function to create a job object to associate with a printable document.

```
OSErr GXNewJob (gxJob *aJob);
```

`aJob`                      On return, a reference to the newly created job object.

*function result*   An error code of type `OSErr`.

### DESCRIPTION

The `GXNewJob` function allocates space for a job object and returns a reference to the job object. You need to call this function each time a user creates a new printable document.

When QuickDraw GX creates a new job object, it contains default values. Specifically, it contains a default format and a default paper type. The default format and default paper type are defined by the default output printer's printer driver. If there is no default output printer's printer driver, the job object uses the format and paper type associated with "Any Printer."

You should call the `GXInstallApplicationOverride` function after you call the `GXNewJob` function to support QuickDraw GX print dialog boxes.

When a user closes a document, you need to dispose of a job object using the `GXDisposeJob` function.

## RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The default paper-type object cannot be located.

## SEE ALSO

Listing 2-1 on page 2-12 shows how to use the `GXNewJob` function to create a job object for a printable document.

The `GXInstallApplicationOverride` function for supporting QuickDraw GX print dialog boxes is described on page 2-71.

To dispose of a job object, see the description of the `GXDisposeJob` function in the next section.

## GXDisposeJob

---

You can use the `GXDisposeJob` function to dispose of a job object associated with a printable document.

```
OSErr GXDisposeJob (gxJob aJob);
```

`aJob`            A reference to the job object to be disposed of.

*function result*   An error code of type `OSErr`.

## DESCRIPTION

You should call the `GXDisposeJob` function when a user closes a printable document and deallocates space for an existing job object. This function returns an error if the specified job object is `nil`.

Before you dispose of a job object, you should call the `GXFlattenJobToHdl` function or the `GXFlattenJob` function to save a job object when a user saves a document.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

Listing 2-9 on page 2-29 shows how to use the `GXDisposeJob` function to dispose of a job object when a user closes a document.

The `GXFlattenJobToHdl` function for saving job objects in a handle is described in the next section. The `GXFlattenJob` function for saving job objects by calling a function is described on page 2-57.

**GXFlattenJobToHdl**

---

You can use the `GXFlattenJobToHdl` function to flatten a job object into a handle.

```
Handle GXFlattenJobToHdl (gxJob aJob, Handle aHandle);
```

<code>aJob</code>	A reference to the job object to be flattened.
<code>aHandle</code>	The handle into which the flattened data is placed.

*function result* The handle into which the flattened data is placed.

**DESCRIPTION**

The `GXFlattenJobToHdl` function provides your application with a mechanism for saving all information associated with a job object in a handle. You should call this function when a user saves a printable document.

You specify a handle in the `aHandle` parameter. QuickDraw GX grows or shrinks the size of the handle you provide to accommodate the size of the job object. You can specify `nil` in this parameter to allow QuickDraw GX to create and return a handle for you.

When you save a printable document, you can write the handle to the file's resource or data fork. You cannot directly modify the contents of this handle.

When a user opens a printable document, you need to unflatten all information associated with a job object using the `UnflattenJobToHdl` function.

If you do not wish to save data in a handle, you can also use the `GXFlattenJob` and `GXUnflattenJob` functions to specify a function to save and restore a job object.

**RESULT CODES**

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

Listing 2-7 on page 2-25 shows how to use the `GXFlattenJobToHdl` function to save a job object.

To unflatten all information associated with a job object, see the `GXUnflattenJobFromHdl` function, which is described on page 2-58.

You can also specify a function to save information associated with a job object by using the `GXFlattenJob` function, which is described in the next section.

**GXFlattenJob**

---

You can use the `GXFlattenJob` function when you want to call a function to flatten a job object.

```
void GXFlattenJob (gxJob aJob,
                  gxPrintingFlattenProc aPrintingFlattenProc,
                  void *aVoid);
```

`aJob`                      A reference to the job object to be flattened.

`aPrintingFlattenProc`      A pointer to a flattening function.

`aVoid`                    A reference variable passed to the flattening function.

**DESCRIPTION**

The `GXFlattenJob` function provides your application with a mechanism for saving all information associated with a job object by specifying a pointer to a flattening function. QuickDraw GX calls your flattening function multiple times as it saves job object-related data to disk.

You specify a pointer to a flattening function in the `aPrintingFlattenProc` parameter of the `GXFlattenJob` function. You may prefer to use the `GXFlattenJob` function (instead of the `GXFlattenJobToHdl` function) because it requires less memory to save portions of job object data to disk than it does to save all the data in a single handle.

When a user opens a printable document, you need to unflatten all information associated with a job object using the `GXUnflattenJob` function.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

Listing 2-8 on page 2-28 shows how to use the `GXFlattenJob` function to save a job object.

An example of a flattening function is described on page 2-76.

To unflatten all information flattened using `GXFlattenJob`, see the `GXUnflattenJob` function, which is described on page 2-59.

To flatten a job to a handle, see the `GXFlattenJobToHdl` function, which is described on page 2-56.

## **GXUnflattenJobFromHdl**

---

You can use the `GXUnflattenJobFromHdl` function to unflatten a job object that you previously flattened using the `GXFlattenJobToHdl` function.

```
gxJob GXUnflattenJobFromHdl (gxJob aJob, Handle aHandle);
```

`aJob`            A reference to the job object into which unflattened data is placed.

`aHandle`        A handle from which the job object is to be read.

*function result*   The unflattened job object.

**DESCRIPTION**

The `GXUnflattenJobFromHdl` function provides your application with a mechanism for retrieving all information associated with a job object from a handle. You should call this function when a user opens a printable document containing a job object that was previously flattened using the `GXFlattenJobToHdl` function.

In the `aJob` parameter, you specify a job object in which to place the unflattened job object data. You can specify `nil` in this parameter to allow QuickDraw GX to create and return a job object for you.

The `aHandle` parameter specifies the handle from which the job object information is read. You previously specified this handle using the `GXFlattenJobToHdl` function.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxFlattenVersionTooNew</code>	An attempt to unflatten a job object that was flattened using a later version of QuickDraw GX. The paper-type object cannot be located.
<code>gxPaperTypeNotFound</code>	
<code>collectionVersionErr</code>	Version of the collection object is not compatible with the current version of the Collection Manager.

**SEE ALSO**

Listing 2-10 on page 2-30 shows how to use the `GXUnflattenJobFromHdl` function to retrieve a job object from a handle.

You specify a handle in which to save a job object using the `GXFlattenJobToHdl` function, which is described on page 2-56.

**GXUnflattenJob**

---

You can use the `GXUnflattenJob` function to unflatten a job object that you previously flattened using the `GXFlattenJob` function.

```
gxJob GXUnflattenJob (gxJob aJob,
                     gxPrintingFlattenProc aPrintingFlattenProc,
                     void *aVoid);
```

`aJob`            A reference to the job object to be unflattened.  
`aPrintingFlattenProc`    A pointer to a flattening function.  
`aVoid`           A reference variable passed to the flattening function.

*function result*    The unflattened job object.

**DESCRIPTION**

The `GXUnflattenJob` function provides your application with a mechanism for retrieving all information associated with a job object by executing an application-supplied function. In the `aPrintingFlattenProc` parameter, you specify a pointer to an unflattening function. QuickDraw GX calls your unflattening function multiple times as it retrieves job object-related data from disk.

In the `aJob` parameter, you specify a job object in which to place the unflattened job object data. You can specify `nil` in this parameter to allow QuickDraw GX to create and return a job object for you.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxFlattenVersionTooNew</code>	An attempt to unflatten a job object that was flattened using a later version of QuickDraw GX.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.
<code>collectionVersionErr</code>	Version of the collection object is not compatible with the current version of the Collection Manager.

**SEE ALSO**

Listing 2-10 on page 2-30 shows an example of how to use the `GXUnflattenJob` function.

You specify a function to save a job object by using the `GXFlattenJob` function, which is described on page 2-57.

**GXUpdateJob**

---

You can use the `GXUpdateJob` function to update the contents of a job object.

```
Boolean GXUpdateJob (gxJob aJob);
```

`aJob`            A reference to the job object whose contents may need to change.

*function result*   A Boolean, which returns `true` if anything actually changed.

**DESCRIPTION**

The `GXUpdateJob` function updates the job object to reflect the current QuickDraw GX environment. You must call this function when your application receives a resume event, indicating that it had been switched out because the user may have changed the characteristics of a printer. For example, the user may have added an extension while the application was switched out.



**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXUpdateJob` function, see “Updating Job Object Information” on page 2-42.

## Printing With QuickDraw GX

---

To support printing from the Finder, your application needs to call the `GXSelectJobOutputPrinter` function to specify an output printer.

When the user requests printing, you should call the `GXGetJobPageRange` function to obtain the user-specified page range.

You call the `GXStartJob` function to begin printing a document. If your application stores each page as a single picture shape, you should use the `GXPrintPage` function to print each page in a document.

You may also choose to use the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions to draw and print data. You should use these functions if your application does not store each page as a single picture shape.

After you have finished calling the `GXPrintPage` or the `GXFinishPage` function (depending on the approach you choose), you call the `GXFinishJob` function to tell QuickDraw GX that the document is ready to be spooled for printing in the background.

The `GXDrawShape` function is described in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

## GXSelectJobOutputPrinter

---

You can use the `GXSelectJobOutputPrinter` function to specify an output printer for a printable document.

```
void GXSelectJobOutputPrinter (gxJob aJob, Str31 printerName);
```

<code>aJob</code>	A reference to the job object for which you are specifying an output printer.
-------------------	---

<code>printerName</code>	The name of the desktop printer.
--------------------------	----------------------------------

**DESCRIPTION**

Your application is responsible for determining the output printer on which to print the document.

For example, when the user selects and prints a document from the Finder, your application needs to respond to the Print Documents ('pdoc') Apple event and then call the `GXSelectJobOutputPrinter` function to specify an output printer on which to print the selected document. The printer name can be obtained by using the Apple event's optional attribute, `keyOptionalKeywordAttr`.

**RESULT CODES**

<code>fnfErr</code>	Printer driver cannot be located.
<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

**SEE ALSO**

Listing 2-16 on page 2-40 shows how to respond to the Print Documents Apple event and use the `GXSelectJobOutputPrinter` function to specify an output printer.

**GXGetJobPageRange**

---

You can use the `GXGetJobPageRange` function to obtain a user-specified page range.

```
void GXGetJobPageRange (gxJob aJob, long *firstPage,
                        long *lastPage);
```

<code>aJob</code>	A reference to the job object for which to retrieve the page range.
<code>firstPage</code>	On return, the first page the user wants to print.
<code>lastPage</code>	On return, the last page the user wants to print.

**DESCRIPTION**

When the user requests printing, you should call the `GXGetJobPageRange` function to obtain the user-specified page range. The user specifies a page range in the Print dialog box.

You can set the `firstPage` parameter or the `lastPage` parameter to `nil` to ignore the result.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>collectionItemNotFoundErr</code>	The collection object item cannot be located.

**SEE ALSO**

Listing 2-5 on page 2-21 and Listing 2-6 on page 2-23 show how to use the `GXGetJobPageRange` function to obtain the user-specified page range.

**GXStartJob**

---

You can use the `GXStartJob` function to initiate printing when a user wants to print a document.

```
void GXStartJob (gxJob aJob, StringPtr docName, long pageCount);
```

<code>aJob</code>	A reference to the job object of the print job to print.
<code>docName</code>	The name of the document to print.
<code>pageCount</code>	The number of pages to print.

**DESCRIPTION**

You use the `GXStartJob` function to begin printing a document. In the `aJob` parameter, you specify the job object associated with the document to print. In the `docName` parameter, you specify the name of the user's document. You can set this parameter to `nil` to use the default document name.

In the `pageCount` parameter, you specify the total number of pages the user chose to print or pass 0 if the page count is unknown. You can call the `GXGetJobPageRange` function to obtain the page range. In response to the `GXStartJob` call, QuickDraw GX displays the current page and the print job's page count, if it is known, in the Status dialog box.

**SPECIAL CONSIDERATIONS**

Immediately after you call the `GXStartJob` function, you should check for errors by calling the `GXGetJobError` function. Only if no errors are returned should you call the `GXFinishJob` function.

**RESULT CODES**

<code>gxPrUserAbortErr</code>	The user has canceled printing.
<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

Listing 2-5 on page 2-21 and Listing 2-6 on page 2-23 show how to use the `GXStartJob` function to begin printing a document.

For information about the `GXGetJobPageRange` function, see the previous section.

The `GXGetJobError` function is described on page 2-52. The `GXFinishJob` function is described on page 2-65.

**GXPrintPage**

---

You can use the `GXPrintPage` function to print a page in a document if your application stores each page as a single picture shape.

```
void GXPrintPage (gxJob aJob, long pageNumber, gxFormat aFormat,
                  gxShape aPage);
```

<code>aJob</code>	A reference to the job object whose page you want to print.
<code>pageNumber</code>	The page number for the page.
<code>aFormat</code>	A reference to the format object for the page.
<code>aPage</code>	A reference to the picture shape that specifies the output for the page.

**DESCRIPTION**

The `GXPrintPage` function prints a page of a document. In the `aPage` parameter, you specify the picture shape for each page. In the `pageNumber` parameter, you set the page to print. QuickDraw GX compares the specified page number with the page range chosen by the user and spools the page if it is within the page range. If it is not within the range, QuickDraw GX ignores the data.

In the `aFormat` parameter, you specify the format object for the page. You need to provide your own mechanism for associating individual document pages with format objects.

You should loop through each page of a document, calling the `GXPrintPage` function for each page's picture shape. You should check for errors after you print each page and exit the loop if necessary.

If your application does not store each page as a single picture shape, you should use the `GXStartPage`, `GXDrawShape`, and `GXFinishPage` functions to print the page.

**RESULT CODES**

<code>gxPrUserAbortErr</code>	The user has canceled printing.
<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

Listing 2-5 on page 2-21 shows how to use the `GXPrintPage` function to print each page of a document.

Picture shapes are discussed in *Inside Macintosh: QuickDraw GX Graphics*.

The `GXStartPage` function is described on page 2-66. The `GXFinishPage` function is described on page 2-67. The `GXDrawShape` function is described in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

In addition to the result codes listed above, you may also receive errors that can occur while flattening graphics objects during spooling. For more information about the spooling phase of printing, see the chapter “Introduction to Printing With QuickDraw GX” in this book. Flattening graphics objects is described in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

**GXFinishJob**

---

You can use the `GXFinishJob` function to notify QuickDraw GX that printing is complete.

```
void GXFinishJob (gxJob aJob);
```

`aJob`            A reference to the job object being printed.

**DESCRIPTION**

The `GXFinishJob` function completes the application phase of printing. You should call this function after you have called the `GXPrintPage` function to print each page in a document.

**SPECIAL CONSIDERATIONS**

You should only call the `GXFinishJob` function if the `GXStartJob` function doesn't return errors.

## RESULT CODES

<code>gxPrUserAbortErr</code>	The user has canceled printing.
<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.

In addition to the result codes listed above, you may also receive errors that can occur while flattening graphics objects during spooling. Flattening graphics objects is described in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

## SEE ALSO

Listing 2-5 on page 2-21 and Listing 2-6 on page 2-23 show how to use the `GXFinishJob` function to tell QuickDraw GX that the document is ready to be queued for printing in the background.

The `GXStartJob` function is described on page 2-63.

Phases of printing are described in the chapter “Introduction to Printing With QuickDraw GX” in this book.

**GXStartPage**

---

You can use the `GXStartPage` function to print each page in a document if your application does not store each page as a single picture shape.

```
Boolean GXStartPage (gxJob aJob, long pageNumber,
                    gxFormat aFormat, long numViewPorts,
                    gxViewPort *viewPortList);
```

<code>aJob</code>	A reference to the job object being printed.
<code>pageNumber</code>	The page number of the page being printed.
<code>aFormat</code>	A reference to the format object for the page.
<code>numViewPorts</code>	The number of view ports contained in the <code>viewPortList</code> parameter.
<code>viewPortList</code>	A pointer to the list of references to view ports to use to capture shapes.

*function result* Returns `true` if the page you specify in the `pageNumber` parameter is within the user-specified page range, `false` if the page you specify is not.

## DESCRIPTION

You use the `GXStartPage` function to start printing the shapes drawn with `GXDrawShape`. You call the `GXStartPage` function after you call the `GXStartJob` function.

In the `GXStartPage` function, you specify in the `pageNumber` parameter the page number of the page to print. QuickDraw GX compares the specified page number with the page range. The `GXStartPage` function returns `true` if the page you specify is within the user-specified page range, and returns `false` if it is not. You can call the `GXGetJobPageRange` function to determine the range of pages.

In the `viewPortList` parameter, you specify the view ports to use to capture shapes. The part of the shape that is drawn through a view port is printed. In the `numViewPorts` parameter, you specify the number of view ports in the `viewPortList` parameter.

#### SPECIAL CONSIDERATIONS

After you finish calling the `GXStartPage` function, you should immediately check for errors using the `GXGetJobError` function. Only if no errors are returned should you draw the page's shapes and call the `GXFinishPage` function.

#### RESULT CODES

`gxPrUserAbortErr`  
`gxSegmentLoadFailedErr`

The user has canceled printing.  
 A required code segment could not be found, or there was not enough memory to load it.

#### SEE ALSO

Listing 2-6 on page 2-23 shows how to use the `GXStartPage` function to print each page of a document.

The `GXDrawShape` function is discussed in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

View port objects are discussed in the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

The `GXFinishPage` function is described in the next section. The `GXGetJobError` function is described on page 2-52. The `GXGetJobPageRange` function is described on page 2-62.

## GXFinishPage

---

You can use the `GXFinishPage` function to notify QuickDraw GX that you have finished capturing shapes for the page.

```
void GXFinishPage (gxJob aJob);
```

`aJob`                      A reference to the job object being printed.

**DESCRIPTION**

You should call the `GXFinishPage` function after you have finished drawing the data for a page using the `GXDrawShape` function. In the `aJob` parameter, you specify the job object being printed.

After you call the `GXFinishPage` function for the final page to be printed, call the `GXFinishJob` function to notify QuickDraw GX that printing is complete.

**SPECIAL CONSIDERATIONS**

You should only call the `GXFinishPage` function if the `GXStartPage` function doesn't return errors.

**RESULT CODES**

<code>gxPrUserAbortErr</code>	The user has canceled printing.
<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.

In addition to the following result codes, you may also receive errors that can occur while flattening graphics objects during spooling. Flattening graphics objects is described in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

**SEE ALSO**

Listing 2-6 on page 2-23 shows how to use the `GXFinishPage` function to tell QuickDraw GX that you have finished capturing shapes.

The `GXStartPage` function is described in the previous section.

The `GXDrawShape` function is discussed in the shape objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

## Obtaining Information on Printing-Related Objects

---

QuickDraw GX functions allow you to obtain basic information about the job object and format objects associated with a printable document. Although a document can contain multiple format objects, all documents contain at least one format object, called the default format.

When a user wants to print a document, you should call the `GXGetJobFormat` function to access the format objects associated with a particular job object.

You can specify a format object in the `GXGetFormatJob` function to obtain the job object that references this format object.

You use the `GXGetFormatDimensions` function to obtain the dimensions information from a format object. The information includes the physical dimensions of the paper (the paper size) and the printable area within these dimensions (the page size) after scaling and orientation have been applied.



## GXGetJobFormat

---

You can use the `GXGetJobFormat` function to obtain the format objects associated with a job object.

```
gxFormat GXGetJobFormat (gxJob aJob, long whichFormat);
```

`aJob`                    A reference to the job object whose format object you wish to obtain.

`whichFormat`            The index of the format object to retrieve.

*function result*    A reference to a format object.

### DESCRIPTION

The `GXGetJobFormat` function allows you to obtain a format object from the job object specified in the `aJob` parameter. The `whichFormat` parameter specifies the format object to return. You can set this parameter to 1 to obtain the default format. The default format is defined by the formatting printer.

### RESULT CODES

`gxSegmentLoadFailedErr`    A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

Listing 2-5 on page 2-21 and Listing 2-6 on page 2-23 show how to use the `GXGetJobFormat` function to obtain the default format when a user wants to print a document.

Manipulating format objects is described in the chapter “Page Formatting and Dialog Box Customization” in this book.

## GXGetFormatJob

---

You can use the `GXGetFormatJob` function to obtain the job object associated with a format object.

```
gxJob GXGetFormatJob (gxFormat aFormat);
```

`aFormat`                    A reference to the format object whose job object you wish to obtain.

*function result*    A reference to a job object.

**DESCRIPTION**

In the `GXGetFormatJob` function, you specify a format object whose job object you want to obtain. You specify the format object in the `aFormat` parameter. You should call this function when you have a format object reference and you want to obtain the reference of the job object associated with it.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**GXGetFormatDimensions**

---

You can use the `GXGetFormatDimensions` function to obtain the page size and paper size associated with a format object.

```
void GXGetFormatDimensions (gxFormat aFormat,
                           gxRectangle *pageSize,
                           gxRectangle *paperSize);
```

<code>aFormat</code>	A reference to the format object whose dimensions you wish to obtain.
<code>pageSize</code>	On return, the imageable area—the area inside the margins where shapes may be drawn.
<code>paperSize</code>	On return, the physical dimensions of the paper.

**DESCRIPTION**

The `GXGetFormatDimensions` function returns a page size and paper size associated with a format object, after scaling and orientation have been applied. This function provides your application with boundary information that is useful for setting up margins for the drawing areas in your application. It is also useful for setting up rulers in your application to display to users.

You can specify `nil` in either the `pageSize` or `paperSize` parameters if you are interested in only one of the values.

The page size is anchored at location (0.0, 0.0), regardless of orientation or scaling. The paper size is outset from the page size, and the coordinates for the top-left corner of the paper are negative. Because the page coordinates are zero-based, you can start drawing at (0.0, 0.0) without regard for the paper size.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For a description of format object mapping and how it affects the dimensions property, see the chapter “Page Formatting and Dialog Box Customization” in this book.

## Displaying the Page Setup and Print Dialog Boxes

---

To support QuickDraw GX print dialog boxes, your application must override the `gxPrintingEvent` message by installing an override function with the `GXInstallApplicationOverride` function.

When the user chooses the Page Setup menu command from the File menu, you call the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box.

When the user chooses the Print menu command from the File menu, you call the `GXJobPrintDialog` function to display the Print dialog box.

## GXInstallApplicationOverride

---

You can use the `GXInstallApplicationOverride` function to override messages QuickDraw GX sends to your application.

```
void GXInstallApplicationOverride (gxJob aJob, short messageID,
                                void *override);
```

`aJob`            A reference to the job object into which to install the override.

`messageID`      The ID of the message to override.

`override`        A pointer to a function with which to override a message.

**DESCRIPTION**

You can use the `GXInstallApplicationOverride` function to specify a function that is called in response to the message specified in the `messageID` parameter. For example, you can override the `gxPrintingEvent` message that QuickDraw GX sends to your application each time it receives an event by specifying a function to call in the `override` parameter.

You specify a pointer to an override function in the `override` parameter. Set this parameter to `nil` to remove your application's override of a message.

**RESULT CODES**

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

Listing 2-1 on page 2-12 shows how to override the `gxPrintingEvent` message using the `GXInstallApplicationOverride` function.

Supporting QuickDraw GX dialog boxes is discussed in “Supporting QuickDraw GX Print Dialog Boxes,” which begins on page 2-17.

**GXJobDefaultFormatDialog**

---

You can use the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box.

```
gxDialogResult GXJobDefaultFormatDialog (gxJob aJob,
                                         gxEditMenuRecord *anEditMenuRecord);
```

`aJob`                      A reference to the job object whose default format you are allowing the user to modify.

`anEditMenuRecord`      A pointer to the Edit menu structure.

*function result*      The user's response to the dialog box.

**DESCRIPTION**

After you use the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box, the user can specify formatting information for the default format. For example, the user can specify the paper size, orientation, and the default formatting printer.

In the `anEditMenuRecord` parameter you specify an Edit menu structure to support the standard editing operations of cut, copy, paste, and clear in dialog boxes.

The function returns `gxOKSelected` if Format is chosen or `gxCancelSelected` if Cancel is chosen.

If an error occurs, the function returns `gxCancelSelected`. Call the `GXGetJobError` function to determine which error occurred.

This function causes QuickDraw GX to send the `gxJobDefaultFormatDialog` message, which you can override to customize the Page Setup dialog box.

Note that QuickDraw GX stores a user's responses to some dialog items in the Page Setup dialog box in a format collection.

## RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

## SEE ALSO

Listing 2-14 on page 2-36 shows how to use the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box.

The Edit menu structure is described on page 2-48.

The dialog box result enumeration is described on page 2-48.

The format collection is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.

## GXJobPrintDialog

---

You can use the `GXJobPrintDialog` function to display the Print dialog box when the user chooses the Print menu command from the File menu.

```
gxDialogResult GXJobPrintDialog (gxJob aJob,
                                gxEditMenuRecord *anEditMenuRecord);
```

`aJob`                      A reference to the job object whose print settings you are allowing the user to modify.

`anEditMenuRecord`      A pointer to the Edit menu structure.

*function result*      The user’s response to the dialog box.

## DESCRIPTION

After you use the `GXJobPrintDialog` function to display the Print dialog box, the user can specify information related to actual printing of the document. For example, the user can specify the printer, print quality, number of copies to print, page range, automatic or manual paper feed, and whether a document should be output to a printer or a file.

A user must select an output printer in the Print dialog box regardless of the formatting printer specified in the Page Setup dialog box. The output printer does not need to be in the same device class as the printer for which the document is formatted.

## Core Printing Features

In the `anEditMenuRecord` parameter you specify an Edit menu structure. Your application specifies the location of the Edit menu and its menu items in the Edit menu structure.

The function returns `gxOKSelected` if Print is chosen or `gxCancelSelected` if Cancel is chosen.

If an error occurs, the function returns `gxCancelSelected`. Call the `GXGetJobError` function to determine which error occurred.

This function causes QuickDraw GX to send the `gxJobPrintDialog` message, which you can override to customize the Print dialog box.

QuickDraw GX stores a user's responses to some items in the Print dialog box in the job collection.

## RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## SEE ALSO

Listing 2-15 on page 2-39 shows how to use the `GXJobPrintDialog` function to display the Print dialog box.

The Edit menu structure is described on page 2-48.

The dialog box result enumeration is described on page 2-48.

The job collection is discussed in the chapter “Page Formatting and Dialog Box Customization” in this book.

## Converting a Print Record

---

QuickDraw GX allows documents originally created to print with the Macintosh Printing Manager to be printed by applications that support QuickDraw GX. Before a user can print these documents, you must convert the document's print record information into a job object using the `GXConvertPrintRecord` function.

## GXConvertPrintRecord

---

You can use the `GXConvertPrintRecord` function to translate a print record into a job object.

```
void GXConvertPrintRecord (gxJob aJob, THPrint aPrint);
```

`aJob`            A reference to the job object to receive the converted data.

`aPrint`          The print record to be converted.

### DESCRIPTION

QuickDraw GX copies contents of the specified print record into the specified job object. Before you call the `GXConvertPrintRecord` function, you must first allocate space for the job object using the `GXNewJob` function. QuickDraw GX attempts to preserve as much print record information as possible.

In addition to converting the print record, you must also translate QuickDraw data by calling the QuickDraw GX Translator functions, `GXInstallQDTranslator` and `GXRemoveQDTranslator`, or by calling the `GXConvertPICTToShape` function.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

### SEE ALSO

Listing 2-18 on page 2-45 shows how to use the `GXConvertPrintRecord` function to convert a print record into a job object.

The `GXNewJob` function is described on page 2-54.

The QuickDraw GX Translator functions, `GXInstallQDTranslator` and `GXRemoveQDTranslator`, are discussed in the environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

For information about the `GXConvertPICTToShape` function, see the environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Application-Defined Functions

---

The following sections describe application-defined functions that implement message overrides and application-defined functions that flatten or unflatten job objects.

### Message Override Functions

---

The `GXPrintingEvent` function specifies the declaration for a function that you must provide in order to respond to `gxPrintingEvent` messages.

### GXPrintingEvent

---

You must install an override function that QuickDraw GX invokes in response to the `gxPrintingEvent` message. Your override must match the following formal declaration:

```
OSErr MyPrintingEvent (EventRecord *anEventRecord,
                      Boolean filterEvent);
```

`anEventRecord`

A pointer to an event that occurred in a print dialog box.

`filterEvent`

A Boolean value that is `true` if the event needs to be filtered, and `false` if not.

*function result* An error code. The value `noErr` indicates that the operation was successful.

#### DESCRIPTION

QuickDraw GX sends the `gxPrintingEvent` message whenever a specific event occurs in one of the print dialog boxes that is displayed for printing. You can override the `gxPrintingEvent` message to handle events, such as window update events, that occur during display of print dialog boxes. You cannot name your function `GXPrintingEvent`.

The default implementation of this message does nothing. You must override this message to correctly support print dialog boxes.

The `anEventRecord` parameter is a pointer to the event record. The event record contains information about what type of event occurred (a mouse-down, update, or key-down event, for example) and contains additional information associated with the event (for example, for a key-down event, the Event Manager also reports which key was pressed).



**SPECIAL CONSIDERATIONS**

You never send the `gxPrintingEvent` message yourself.

You typically create a total override of the `gxPrintingEvent` message.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

**SEE ALSO**

Overriding the `gxPrintingEvent` message is described in “Supporting QuickDraw GX Print Dialog Boxes,” which begins on page 2-17.

The `GXInstallApplicationOverride` function is described on page 2-71.

The Event Manager, the `EventRecord` data type, and the `DialogSelect` function are discussed in the chapter “Event Manager” in *Inside Macintosh: Macintosh Toolbox Essentials*.

## Flattening and Unflattening Functions for Job Objects

---

When a user saves or opens a printable document, you need to save or retrieve its corresponding job object. To save a job object, you can flatten it using the `GXFlattenJob` function. To retrieve a job object, you can unflatten it using the `GXUnflattenJob` function. In each of these functions you must provide a pointer to an application-supplied flattening or unflattening function, as appropriate. The following sections describe these flattening and unflattening functions.

### MyFlattenFunction

---

To save a job object when a user saves a printable document, provide a pointer to an application-supplied flattening function in the `GXFlattenJob` function. The application-supplied function must match the following declaration. For example, this is how you should declare the function if you were to name it `MyFlattenFunction`:

```
OSErr MyFlattenFunction (long size, void *data, void *refCon);
```

<code>size</code>	The size of the segment (in bytes) to write.
<code>data</code>	A pointer to job object data to flatten.
<code>refCon</code>	A pointer to a reference constant for application-specific information.

*function result* An error code of type `OSErr`.

**DESCRIPTION**

When you use the `GXFlattenJob` function, QuickDraw GX calls your flattening function multiple times as it flattens job object data to disk. Each time it calls your function, the function should write the next segment of the job object until the entire job object is saved. You can use the `refCon` parameter to hold the file reference number of the file containing the data to flatten. You can return any `OSErr` value.

**SEE ALSO**

Listing 2-8 on page 2-28 shows how to use a flattening function.

The `GXFlattenJob` function is described on page 2-57.

To retrieve a job object that has been flattened, see the next section.

## MyUnflattenFunction

---

To retrieve a job object when a document is opened, you can call the `GXUnflattenJob` function and provide a pointer to the application-supplied unflattening function you want to use. The application-supplied function must match the following declaration. For example, this is how you should declare the function if you were to name it `MyUnflattenFunction`:

```
OSErr MyUnflattenFunction (long size, void *data, void *refCon);
```

<code>size</code>	The size of the segment (in bytes) to read.
<code>data</code>	A pointer to job object data to unflatten.
<code>refCon</code>	A pointer to a reference constant for application-specific information.

*function result* An error code of type `OSErr`.

**DESCRIPTION**

When you use the `GXUnflattenJob` function, QuickDraw GX calls your unflattening function multiple times as the unflattening function retrieves the job object data from disk. It continues to call your function until the entire job object is retrieved. You can use the `refCon` parameter to hold the file reference number of the file containing the data to unflatten. You can return any `OSErr` value.

**SEE ALSO**

Listing 2-11 on page 2-32 shows how to use an unflattening function.

The `GXUnflattenJob` function is described on page 2-59.

## Summary of Core Printing Features

---

### Constants and Data Types

---

#### Gestalt Selectors for Printing

```
#define gestaltGXPrintingMgrVersion 'pmgr'
#define gestaltGXVersion           'qdgx'
```

#### QuickDraw GX Printing-Related Objects

```
/* printing-related object structures */
typedef struct gxPrivateJobRecord *gxJob;           /* job object structure */
typedef struct gxPrivatePrinterRecord *gxPrinter;   /* printer object */
                                                    /* structure */
typedef struct gxPrivateFormatRecord *gxFormat;     /* format object */
                                                    /* structure */
typedef struct gxPrivatePaperTypeRecord *gxPaperType; /* paper-type object */
                                                    /* structure */
typedef struct gxPrivatePrintFileRecord *gxPrintFile; /* print file object */
                                                    /* structure */
typedef struct PrivateCollectionRecord *Collection; /* collection object */
                                                    /* structure */
```

#### Edit Menu Record Structure

```
typedef struct {
    short editMenuID; /* location of Edit menu and its menu items */
    short cutItem;    /* resource ID of the Edit menu */
    short cutItem;    /* location of the cut menu item */
    short copyItem;   /* location of the copy menu item */
    short pasteItem;  /* location of the paste menu item */
    short clearItem;  /* location of the clear menu item */
    short undoItem;   /* location of the undo menu item */
} gxEditMenuRecord;
```

#### Dialog Box Results

```
typedef long gxDialogResult; /* dialog result data type */

/* dialog box result enumeration */
```

```
enum {
    gxCancelSelected = (gxDialogResult) 0, /* user canceled dialog box */
    gxOKSelected      = (gxDialogResult) 1, /* user confirmed dialog box */
    gxRevertSelected  = (gxDialogResult) 2 /* user chose Remove from
                                           the Custom Page Setup
                                           dialog box */
};
```

## Functions

---

### Initializing and Terminating QuickDraw GX Printing Features

```
OSErr GXInitPrinting      (void);
OSErr GXExitPrinting      (void);
```

### Handling Errors

```
OSErr GXGetJobError       (gxJob aJob);
void GXSetJobError        (gxJob aJob, OSErr anError);
```

### Creating and Managing Job Objects

```
OSErr GXNewJob            (gxJob *aJob);
OSErr GXDisposeJob        (gxJob aJob);
Handle GXFlattenJobToHdl  (gxJob aJob, Handle aHandle);
void GXFlattenJob          (gxJob aJob,
                           gxPrintingFlattenProc aPrintingFlattenProc,
                           void *aVoid);
gxJob GXUnflattenJobFromHdl (gxJob aJob, Handle aHandle);
gxJob GXUnflattenJob      (gxJob aJob,
                           gxPrintingFlattenProc aPrintingFlattenProc,
                           void *aVoid);
Boolean GXUpdateJob       (gxJob aJob);
```

### Printing With QuickDraw GX

```
void GXSelectJobOutputPrinter (gxJob aJob, Str31 printerName);
void GXGetJobPageRange        (gxJob aJob, long *firstPage, long *lastPage);
void GXStartJob               (gxJob aJob, StringPtr docName, long pageCount);
void GXPrintPage              (gxJob aJob, long pageNumber, gxFormat aFormat,
                              gxShape aPage);
```

```
void GXFinishJob          (gxJob aJob);
Boolean GXStartPage      (gxJob aJob, long pageNumber, gxFormat aFormat,
                          long numViewPorts, gxViewPort *viewPortList);
void GXFinishPage        (gxJob aJob);
```

#### Obtaining Information on Printing-Related Objects

```
gxFormat GXGetJobFormat   (gxJob aJob, long whichFormat);
gxJob GXGetFormatJob      (gxFormat aFormat);
void GXGetFormatDimensions (gxFormat aFormat, gxRectangle *pageSize,
                          gxRectangle *paperSize);
```

#### Displaying the Page Setup and Print Dialog Boxes

```
void GXInstallApplicationOverride
                          (gxJob, aJob, short messageID,
                          void *override);

gxDialogResult GXJobDefaultFormatDialog
                          (gxJob aJob,
                          gxEditMenuRecord *anEditMenuRecord);

gxDialogResult GXJobPrintDialog
                          (gxJob aJob,
                          gxEditMenuRecord *anEditMenuRecord);
```

#### Converting a Print Record

```
void GXConvertPrintRecord (gxJob aJob, THPrint aPrint);
```

#### Application-Defined Functions

---

##### Message Override Functions

```
OSErr GXPrintingEvent    (EventRecord *anEventRecord,
                          Boolean filterEvent);
```

##### Flattening and Unflattening Functions for Job Objects

```
OSErr MyFlattenFunction   (long size, void *data, void *refCon);
OSErr MyUnflattenFunction (long size, void *data, void *refCon);
```



# Page Formatting and Dialog Box Customization

---

## Contents

About Page Formatting and Dialog Box Customization	3-6
About Collection Objects	3-7
Collection Tag IDs and Item IDs	3-7
Item Structures	3-8
Categories of Collection Items	3-9
The Job Collection	3-10
The Format Collection	3-12
The Paper-Type Collection	3-14
About Page Formatting	3-15
Manipulating Format Objects	3-16
Mapping for Format Objects	3-18
Forms and Format Objects	3-20
Halftones and Format Collections	3-21
Dialog Box Customization	3-22
The Dialog Box Panel Resource	3-24
Responding to Panel Events	3-25
Automating Panel Events	3-25
Using Printing-Related Collection Objects	3-27
Accessing Data From a Collection Object	3-28
Using a Collection to Implement the Print One Copy Menu Item	3-29
Replacing Items in Collections	3-31
Specifying Page Ranges in the Job Collection	3-33
Using Format Objects and Collection Items to Format Pages	3-39
Creating a Format Object for a Page in a Document	3-40
Sharing Formats for Document Pages	3-44
Disposing of a Format Object for a Page in a Document	3-47
Using Forms With Format Objects	3-50

Storing Halftone Information in a Format Collection	3-52
Copying a Format Object for Use in Other Documents	3-54
Obtaining the Mapping From a Format Object	3-57
Obtaining a Paper-Type Object Associated With a Format	3-57
Scanning Through a Job's Format Objects	3-59
Associating Format Objects With Document Pages	3-61
Customizing QuickDraw GX Dialog Boxes	3-66
Adding Panels to Dialog Boxes	3-67
Setting Up Dialog Box Resources	3-70
Parsing Page Ranges	3-73
Page Formatting and Dialog Box Customization Reference	3-75
Constants for Loop Status Information	3-76
Constants for Collection Item Categories and Tag IDs	3-76
Collection Item Categories	3-76
Collection Tag ID	3-77
Constants and Data Types for Job Collection Items	3-78
Print-Job Information	3-78
Collation Information	3-80
Copies Information	3-81
Page-Range Information	3-81
Quality Information	3-83
File-Destination Information	3-83
File-Location Information	3-84
File-Format Information	3-84
File-Fonts Information	3-85
Paper-Feed Information	3-85
Manual-Feed Information	3-86
Standard Mapping Information	3-86
Special Mapping Information	3-87
Tray-Mapping Information	3-88
Print-Panel Information	3-88
Format-Panel Information	3-88
Paper-Mapping Information	3-89
Translated-Document Information	3-89
Constants and Data Types for Format Collection Items	3-89
Orientation Information	3-89
Scaling Information	3-91
Direct-Mode Information	3-91
Format-Halftone Information	3-92
Page-Inversion Information	3-92
Horizontal Page-Flip Information	3-93
Vertical Page-Flip Information	3-93
Precise-Bitmap Information	3-93
Paper-Type Lock Information	3-94
Constants and Data Types for Paper-Type Collection Items	3-94
Base Information	3-94
Creator Information	3-95



Units Information	3-96
Flags Information	3-97
Comment Information	3-97
Panel-Related Constants and Data Types	3-98
The Panel Information Structure	3-98
Panel Events	3-99
Panel Responses	3-100
Panel Event Actions	3-101
The Panel Setup Structure	3-101
Printing Panel Kinds	3-102
Parse Range Results	3-102
Functions	3-103
Creating and Manipulating Format Objects	3-103
GXNewFormat	3-104
GXDisposeFormat	3-104
GXCopyFormat	3-105
GXCloneFormat	3-106
GXCountJobFormats	3-107
GXCountFormatOwners	3-107
GXForEachJobFormatDo	3-108
Manipulating Format Object Properties	3-109
GXGetFormatMapping	3-109
GXGetFormatPaperType	3-110
GXGetFormatForm	3-111
GXSetFormatForm	3-111
GXChangedFormat	3-112
Displaying the Custom Page Setup Dialog Box	3-113
GXFormatDialog	3-113
Working With Panels	3-114
GXSetupDialogPanel	3-114
GXGetJobPanelDimensions	3-115
GXEnableJobScalingPanel	3-116
GXGetMessageHandlerResFile	3-116
Accessing Printing-Related Collection Objects	3-117
GXGetJobCollection	3-117
GXGetFormatCollection	3-118
GXGetPaperTypeCollection	3-118
Application-Defined Functions	3-119
Message Override Functions for Customizing QuickDraw GX Dialog Boxes	3-119
GXJobPrintDialog	3-120
GXJobDefaultFormatDialog	3-121
GXFormatDialog	3-122
GXHandlePanelEvent	3-123
GXFilterPanelEvent	3-124
GXParsePageRange	3-125

## CHAPTER 3

Looping Through Format Objects	3-126
Dialog Box-Related Resources	3-127
The Panel Resource	3-127
The Extended Item List Resource	3-128
Summary of Page Formatting and Dialog Box Customization	3-133

This chapter describes how your application can manipulate the objects that QuickDraw GX uses to format the pages of a document or add panels to QuickDraw GX dialog boxes.

Read the information in this chapter if you want your application to allow users to specify unique formats for the individual pages of a printable document. For example, using QuickDraw GX, your application can allow a user to create and print a single document that consists of an address page on an envelope, a business letter on a sheet of paper in portrait orientation, and a spreadsheet on a sheet of paper in landscape orientation.

You should also read this chapter if you want to add panels to QuickDraw GX print dialog boxes. For example, your application may add a panel that allows the user to specify additional information, such as color-separation for color printing.

Before you begin using QuickDraw GX page formatting and dialog box customization features, you should be familiar with the basic concepts for printing with QuickDraw GX, as described in the chapter, “Introduction to Printing With QuickDraw GX.” You should also be familiar with creating and manipulating a job object, as described in the chapter “Core Printing Features” in this book.

This chapter begins by summarizing what you need to know to support the page formatting and dialog box customization features of QuickDraw GX. Because page formatting and dialog box customization can use collection objects, this topic is introduced first. Page formatting is discussed next because you can use collection items as parameters to specify formatting criteria. Dialog box customization is discussed after the other two topics because you may need to use nondefault dialog boxes to allow the user to set the values of items in a collection object. Keep in mind that any QuickDraw GX print dialog box can be customized, not just the Custom Page Setup dialog box associated with page formatting.

After introducing the basic concepts associated with printing-related collection objects, page formatting, and dialog box customization, this chapter shows you how to

- n access an item in a collection object for use with a dialog box
- n keep track of format objects that are shared by multiple pages of a document
- n create a format object for a page in a document
- n clone a format object for multiple pages in a document
- n dispose of a format object for a page in a document
- n access information associated with a format object
- n display the Custom Page Setup dialog box
- n support special formatting features
- n associate format objects with document pages
- n add panels to QuickDraw GX dialog boxes
- n automate panel information

## About Page Formatting and Dialog Box Customization

---

Page formatting is the ability to format individual pages of a document differently from the default format for the document. The available formats are specified by the printer driver. You specify a format object when you print each page of a document. If you specify the first format in the job object's format list, the default format for the document is used. If you specify another format, it is used to format the page. For more information about printing pages and specifying formats, see the chapter "Core Printing Features" in this book.

Typically, you associate the default format object with each page in the document and let the user choose the pages to format differently from the default. The user can choose the format with the Custom Page Setup menu item of the File menu, which displays the Custom Page Setup dialog box on the user's screen. You are responsible for associating the chosen format with the page. Thus, you need to determine which format objects are in use and save them with the job object when the document is saved. You also need to retrieve them along with the job object when the document is opened. For more information about saving and retrieving these job objects, see "Associating Format Objects With Document Pages" on page 3-61.

The Custom Page Setup dialog box provided by QuickDraw GX allows the user to format a page, remove the format and revert to the default format, change the paper type for the page, and change the page's scale and orientation. You can allow more choices by customizing this dialog box. For example, you can allow the user to specify a halftone to be applied to the page. Because the Custom Page Setup dialog box provided by QuickDraw GX does not provide an option for specifying a page halftone, the printer driver or a printing extension must customize the dialog box, or you must customize the dialog box in the application.

If you customize a dialog box, you typically gather additional information from the user, although you can also customize a dialog box to restrict the user's choices. The additional information is stored in a collection object. In the halftone example, the printer driver stores the possible halftone options in the format collection. You can customize the Page Setup dialog box to allow a halftone to be chosen for the default format, or you can customize the Custom Page Setup dialog box to allow a halftone to be chosen for a particular page.

QuickDraw GX allows you to customize any of the print dialog boxes:

- n The Print dialog box, in which the user specifies parameters for printing the job.
- n The Page Setup dialog box, in which the user specifies default formatting by selecting the formatting printer. This dialog box is also used to specify the default paper type.
- n The Custom Page Setup dialog box, in which the user specifies formatting for a particular page, including the paper type.
- n The Printing Status dialog box, in which the status of the spooling operation is displayed. This dialog box is not usually customized. You may choose, however, to suppress the display of the dialog box under certain conditions.

## About Collection Objects

---

QuickDraw GX supports collection objects to store and to allow your application to store printing-related, formatting, and paper-type information associated with a printable document. Essentially, these collections specify additional information that are not absolutely required to print a job, format a document, or specify the kind of paper. In QuickDraw GX printing, collection objects typically store information you can use to customize dialog boxes. You can access information required by your application from these collection objects, however, whether or not you allow the user a choice in a dialog box. You can also use collection objects to store information that is of use only to your application.

You can use collection objects without customizing dialog boxes. For example, a user may print by dragging the document's icon onto a desktop printer or by choosing the Print One Copy menu item from the File menu. In these cases, your application may need to change the settings in a collection object directly, without user intervention.

You can also store information that is not already provided by QuickDraw GX. For example, as part of using QuickDraw GX page formatting features, your application is responsible for managing the correspondence between format objects and individual pages in a document. Your application can use a format collection item to store this correspondence. Storing correspondence information in a format collection is discussed in "Associating Format Objects With Document Pages," which begins on page 3-61.

## Collection Tag IDs and Item IDs

---

When you add data (referred to as a collection item) to a collection object, the Collection Manager associates the data with a collection tag ID and a collection item tag. Together, the 4-byte collection tag ID and the 4-byte collection item tag ID uniquely identify a collection item within a particular collection object.

### Note

To avoid the confusion between tag objects (which are not related to collection objects at all), collection tags, and collection item tags, this book refers to collection tags as tag IDs and to collection item tags as item IDs. Tags, when used in this book, refer to tag objects. u

QuickDraw GX assigns the `gxPrintingTagID` tag ID to each of its predefined collection items:

```
enum { gxPrintingTagID = -28672 };
```

For each of its collection items, QuickDraw GX defines an item ID, such as `gxCopiesTag` for the collection item that defines the number of copies to print:

```
enum { gxCopiesTag = 'copy' };
```

QuickDraw GX reserves all tag IDs that are negative or less than 127. It also reserves all collection items defined by lowercase characters. For example, you can use your application's registered creator type for the tag ID.

In addition to the collection tag and collection item ID, the Collection Manager allows items to be accessed by index. You can use an index to provide faster access to specific items in a collection or to perform operations on all collection items in a collection object. This index does not uniquely identify an item, however, because adding or removing items can change an item's index number. For information about collection indexes and collection objects in general, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Item Structures

---

A structure defines the form of most collection items. A type definition is associated with each of these structures:

```
struct gxCopiesInfo{
    long copies;
};

typedef struct gxCopiesInfo gxCopiesInfo;
```

For example, you can use `gxCopiesInfo` as both a structure name and a data type definition:

```
gxCopiesInfo myCopies;
struct gxCopiesInfo myCopies;
```

In this book, only the structure definition is presented. Type definitions are only presented when they are not associated with a structure, as in `gxCollectionCategory`, defined in the next section.

## Categories of Collection Items

---

If you add an item to a collection, you need to decide whether the contents will be valid if the output printer or formatting printer changes. You also must decide if the item should persist when the collection is flattened.

QuickDraw GX purges the items that are not valid after the printer driver changes, based on the contents of the collection item's user attribute bits. It also decides which items to flatten based on these bits. For general information about user attribute bits, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

A printer-driver switch occurs whenever a user changes the device class (type of printer) of the output printer or formatting printer associated with a particular document. For example, if the user switches output printers, QuickDraw GX discards the tray-feed information, which specifies the current paper tray, because it may have changed.

QuickDraw GX assigns collection object items into categories based on the contents of the `gxCollectionCategory` user attribute bits, as shown in the following enumeration:

```
typedef short gxCollectionCategory;

enum {
    gxNoCollectionCategory          = (gxCollectionCategory) 0x0000,
    gxOutputDriverCategory          = (gxCollectionCategory) 0x0001,
    gxFormattingDriverCategory      = (gxCollectionCategory) 0x0002,
    gxDriverVolatileCategory        = (gxCollectionCategory) 0x0004,

    gxVolatileOutputDriverCategory =
        gxOutputDriverCategory + gxDriverVolatileCategory,
    gxVolatileFormattingDriverCategory =
        gxFormattingDriverCategory + gxDriverVolatileCategory
};
```


Items in the `gxNoCollectionCategory` category are not purged. Data that is specific to an output printer driver should be grouped in the `gxVolatileOutputDriverCategory` collection item category. Data that is specific to a formatting printer driver should be grouped in the `gxVolatileFormattingDriverCategory` collection item category.

Data that need not be saved when a job is flattened should be grouped in the `gxDriverVolatileCategory` collection item category. You must also clear the `collectionPersistenceBit` attribute bit if you would like to keep the information but do not require it to be saved with the collection.

## The Job Collection

QuickDraw GX primarily stores in a job collection the information contained in the Print dialog box and its General panel, Paper Match panel, and Print Time panel. Panels for the Print dialog box are discussed in the chapter “Introduction to Printing With QuickDraw GX” in this book. QuickDraw GX stores 18 items in a job collection, as shown in Figure 3-1.

**Figure 3-1** The job collection



Job collection
Printjob information
Collation information
Copies information
Page-range information
Quality information
File-destination information
File-location information
File-format information
File-fonts information
Paper-feed information
Manual-feed information
Standard mapping information
Special mapping information
Tray-mapping information
Printpanel information
Formatpanel information
Paper-matching information
Translated-document information

A brief description of each collection item follows. To see how the pieces of data are structured in the collection item, see “Constants and Data Types for Job Collection Items” beginning on page 3-78. Job collection items include the following:

- n **Print-job information.** This collection item describes the job information for the print job. It contains information such as the total number of pages to print, the print job’s priority, designated time to print, and the amount of time in which to keep a print job



in an alert state before cancelling the job. It also contains the first page from which to begin printing and indicates whether the user chose to be alerted before printing begins or after printing is finished. In addition, this property contains the name of the application used to create the printable document, the name of the user's document, and the name of the user associated with the printable document.

- n **Collation information.** This collection item specifies whether document pages should be collated when printed. The user typically specifies whether collation is desired in the Collate Copies checkbox in the Print dialog box.
- n **Copies information.** This collection item contains the number of copies of the document to print. The user specifies the number of copies to print in the Copies field in the Print dialog box.
- n **Page-range information.** This collection item contains the page-range information in the job object as well as data that allows customized or replacement page ranges. It contains the user-specified custom, default, or replacement page-range information from the Print dialog box.
- n **Quality information.** This collection item contains information about the quality mode, such as the default quality mode and the current mode. It also includes the number of quality menu items and an array of quality names (such as "Best") to display in the Quality pop-up menu in the Print dialog box.
- n **File-destination information.** This collection item contains the file-destination information for the job object. It specifies whether the user chose File in the Destination pop-up menu in the Print dialog box.
- n **File-location information.** This collection item contains the file-location information as a `FSSpec` structure. It typically contains the result of a call to `StandardGetFile`, which is used to determine the filename when the user prints to a file.
- n **File-format information.** This collection item contains the name of the file format if the destination of the print job is a file.
- n **File-fonts information.** This collection item specifies whether fonts should be stored as part of the file. If fonts are stored, it specifies whether all fonts are stored or only nonstandard fonts.
- n **Paper-feed information.** This collection item contains the paper-feed information for the job object. It specifies whether the user chose the Automatic or Manual radio button for Paper Feed in the Print dialog box.
- n **Manual-feed information.** This property contains the manual-feed information for the job object. It specifies the number of paper types to manually feed and an array of paper-type names to display.
- n **Standard mapping information.** This collection item specifies whether to use standard mapping information for the print job. The item contains a Boolean value that is `true` if input tray paper matching is to be used.
- n **Special mapping information.** This collection item contains the special mapping information for the job object. It specifies mapping options, such as whether to redirect the pages in a document to a particular paper tray or whether to scale pages or tile pages in a document.

- n **Tray-mapping information.** This collection specifies the tray-mapping information for the job object. It contains the index for a paper tray, which the user typically specifies by selecting a tray from the Paper Match panel of the Print dialog box.
- n **Print-panel information.** This collection item contains the print-panel information for the job object. It specifies the name of the first panel to appear when your application displays the Print dialog box.
- n **Format-panel information.** This collection item contains the format-panel information for the job object. It specifies the name of the first panel to appear when your application displays the Page Setup dialog box.
- n **Paper-mapping information.** This collection item contains the paper-mapping information for the job object. If it is used, it contains a flattened paper-type resource.
- n **Translated-document information.** This collection item contains the translated-document information for the job object. QuickDraw GX provides this information only for documents designed for printing with the Macintosh Printing Manager.

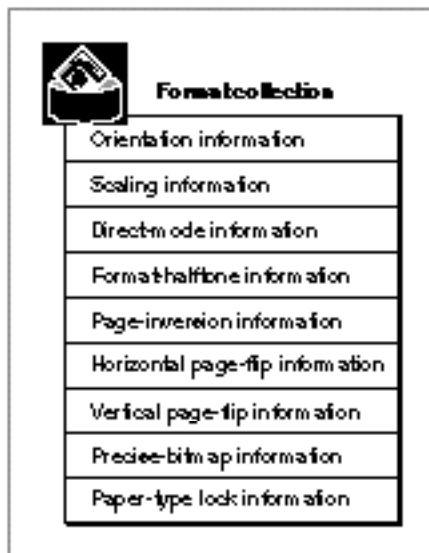
### The Format Collection

---

QuickDraw GX primarily stores information from the Page Setup and Custom Page Setup dialog boxes in a format collection. You need to call the `GXChangedFormat` function each time you change the format collection.

QuickDraw GX stores nine items in a format collection, as shown in Figure 3-2.

**Figure 3-2** The format collection



A brief description of each collection item follows. To see how the pieces of data are structured in the collection item, see “Constants and Data Types for Format Collection Items” beginning on page 3-89. Format collection items include the following:

- n **Orientation information.** This collection item contains the orientation information for the format object. It specifies whether to print a document (or a specific page) in portrait, landscape, or rotated landscape orientation. A user typically specifies orientation for an entire document in the Page Setup dialog box and specifies orientation for an individual page in the Custom Page Setup dialog box.
- n **Scaling information.** This collection item contains the scaling information for the format object. It specifies a document’s horizontal and vertical scaling factors. It also stores the minimum and maximum scaling factors allowed. A user typically specifies scaling for an entire document in the Page Setup dialog box and specifies scaling for an individual page in the Custom Page Setup dialog box.
- n **Direct-mode information.** This collection item contains the direct-mode information for the format object. It specifies whether the user chose the Direct checkbox in the Page Setup dialog box. (This checkbox appears only if the printer driver supports text job format mode printing.) The text job format mode is discussed in the chapter “Advanced Printing Features” in this book.
- n **Format-half-tone information.** This collection item contains the format-half-tone information for the format object. It specifies the total number of half-tone structures that can be used for a specific page and an array of half-tone structures. You can use halftones to render continuous tone images on noncontinuous tone printers if the printer driver or a printing extension supports halftones. For an introduction to halftones, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*. For more information about this collection item, see “Halftones and Format Collections” beginning on page 3-21.
- n **Page-inversion information.** This collection item contains the page-inversion information for the format object. It specifies whether to invert a page before printing.
- n **Horizontal page-flip information.** This collection item contains the horizontal page-flip information for the format object. It specifies whether to horizontally flip the page left to right before printing.
- n **Vertical page-flip information.** This collection item contains the vertical page-flip information for the format object. It specifies whether to vertically flip the page top to bottom before printing.
- n **Precise-bitmap information.** This collection item contains the precise-bitmap information for the format object. It specifies whether to scale a page by 96% on 300-dpi printers.
- n **Paper-type lock information.** This collection item contains the paper-type object lock information for the format object. It indicates whether the format’s paper-type object is locked.

#### Note

The page-inversion information, page-flip information (horizontal and vertical), and precise-bitmap information are used, by default, only by PostScript printer drivers. u

## The Paper-Type Collection

The paper-type collection contains additional information about the paper-type object. QuickDraw GX stores in a paper-type collection five collection items, as shown in Figure 3-3.

**Figure 3-3** The paper-type collection



A brief description of each collection item follows. To see how the pieces of data are structured in the collection item, see “Constants and Data Types for Paper-Type Collection Items” beginning on page 3-94. Paper-type collection items include the following:

- n **Base information.** This collection item contains the base paper type information for the paper-type object, which indicates the source from which the paper type was created. Base types include: unknown, US Letter, US Legal, A4, B5, and tabloid.
- n **Creator information.** This collection item contains the creator information structure for the paper-type object. It specifies the creator type of a paper-type object; for example, 'sypt' for a system paper-type object creator and 'uspt' for a user paper-type object creator.
- n **Units information.** This collection item contains the units information for the paper-type object. Units can be specified in picas, millimeters, and inches.
- n **Flags information.** This collection item contains the flags information for the paper-type object. The flags are bits used to set or clear specific attributes of a paper-type object, such as whether the paper type is the default paper type for this format. For information about paper-type object flags, see “Flags Information” beginning on page 3-97.
- n **Comment information.** This collection item contains the comment information for the paper-type object. It allows a comment to be associated with a paper-type object. You can specify application-specific information in this comment. For example, you may want to store a textual description of the paper-type and its purpose.

## About Page Formatting

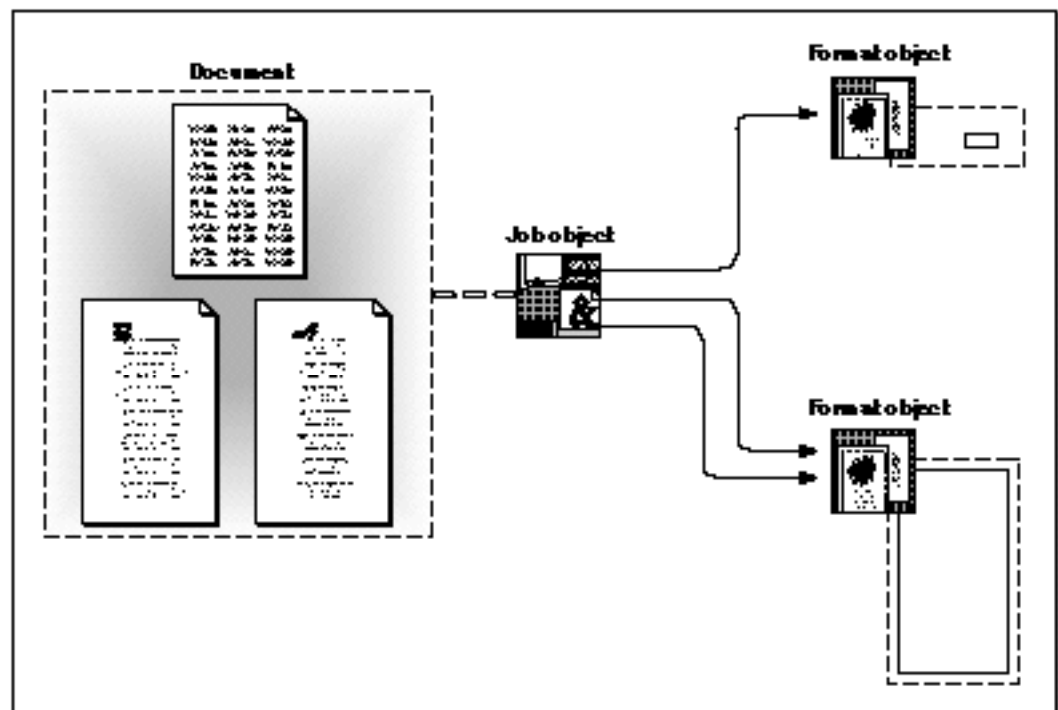
Page formatting allows the user to format specific pages of a document differently from the default formatting for the rest of the document. Using QuickDraw GX page formatting features, your application can

- n allow users to specify unique formats for the individual pages of a document
- n retrieve a format object's mapping
- n attach a form to a format object as a backdrop to each page
- n create documents that contain page-specific halftone information
- n copy a format object for use in other documents

For example, using page-formatting features, a mail-merge application may automatically generate a document in which the first page consists of a template in which a user can enter addresses and the rest of the document consists of blank sheets in which a user can add text.

Figure 3-4 shows a document that is composed of a two-page letter and many address labels. The job object references two format objects, one for either page of the letter and the other for the address label.

**Figure 3-4** A three page document and its corresponding job and format objects



Manipulating format objects is described in the next section. Information on accessing a format object's mapping is discussed in "Mapping for Format Objects" beginning on page 3-18. Information on attaching a form to a format object is discussed in "Forms and Format Objects" beginning on page 3-20. Information on halftones is discussed in "Halftones and Format Collections" beginning on page 3-21.

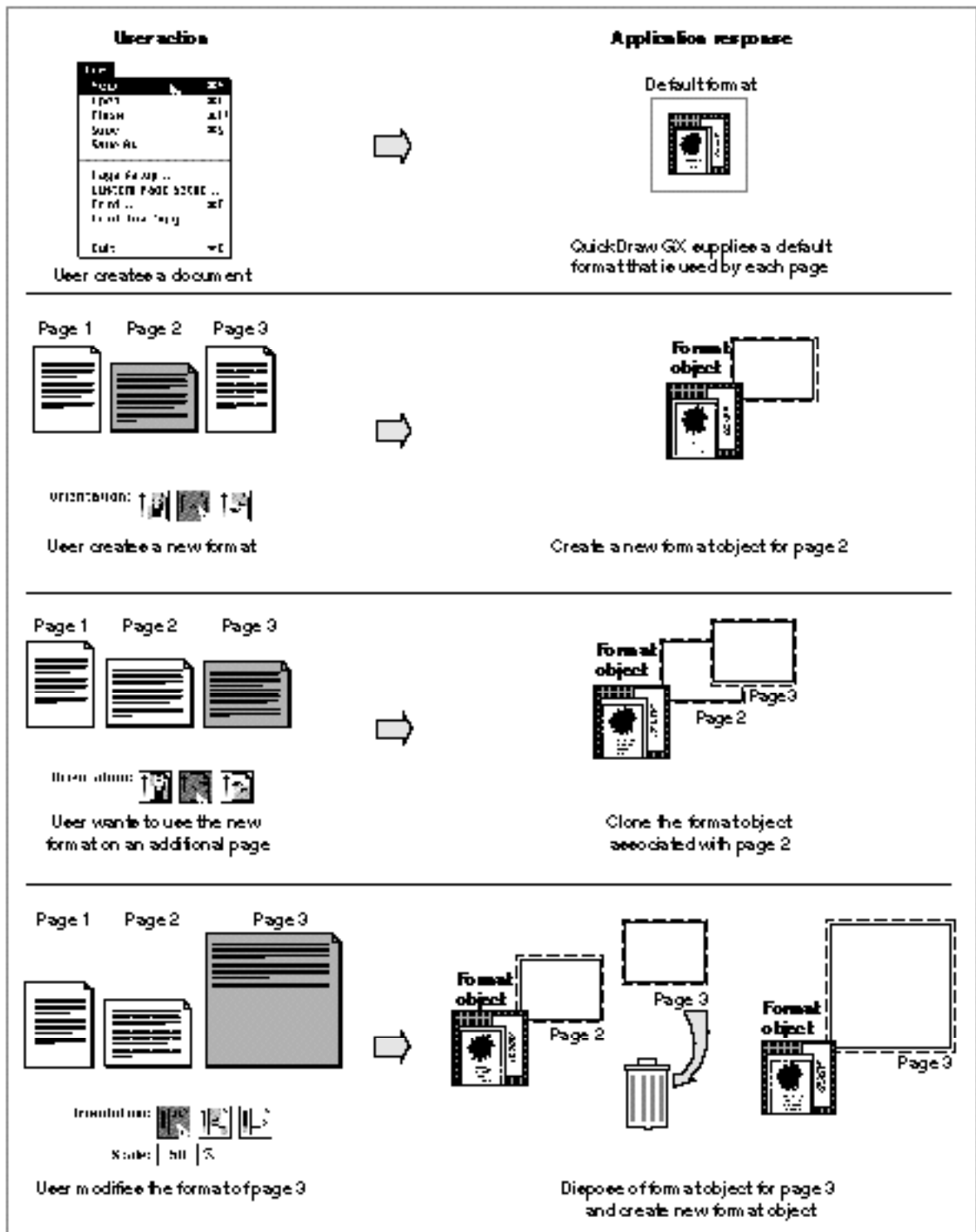
## Manipulating Format Objects

---

A format object contains the basic information that your application needs to display a single page or a set of pages. Generally, you work with format objects when a user

- n creates a new format using the Custom Page Setup dialog box
- n wants to use a format in several pages of a document
- n modifies a format that is shared by other pages in the same document
- n saves or opens a document

Figure 3-5 shows how you manipulate format objects in response to the first three actions.

**Figure 3-5** Manipulating the format object in response to user actions

When a user creates a new format through the Custom Page Setup dialog box, you need to create a new format object. Creating a new format object is discussed in “Creating a Format Object for a Page in a Document,” which begins on page 3-40.

Each format object you create has an associated owner count. The owner count indicates the number of times that a format object is shared. When a user creates a new format through the Custom Page Setup dialog box, you need to create a new format object with the `GXNewFormat` function. This function sets the owner count of a format object to 1.

When a user wants to use the new format to format another page the same way, you need to increment the format object’s owner count. You use the `GXCloneFormat` function to increment the owner count of a format object by 1. Cloning a format object is discussed in “Sharing Formats for Document Pages,” which begins on page 3-44.

When a user modifies a format object that is also shared by other pages, you need to dispose of its corresponding format object and create a new one. The `GXDisposeFormat` function decrements the owner count of a format object by 1. Disposing of a format object is discussed in “Disposing of a Format Object for a Page in a Document,” which begins on page 3-47.

To obtain the current owner count of a format object, you use the `GXCountFormatOwners` function. For more information about this function, see the description of `GXCountFormatOwners` on page 3-107.

You also must create a correspondence between the format and the page. Typically, you keep the correspondence in the format collection. You must save the correspondences when the job is flattened and retrieve them when the job is unflattened. For an example, see “Associating Format Objects With Document Pages” beginning on page 3-61.

## Mapping for Format Objects

---

A format object’s mapping is a mathematical representation of the format object’s settings. These settings include the paper size, page size, orientation, and scaling. The paper size and page size are set when you create the format object.

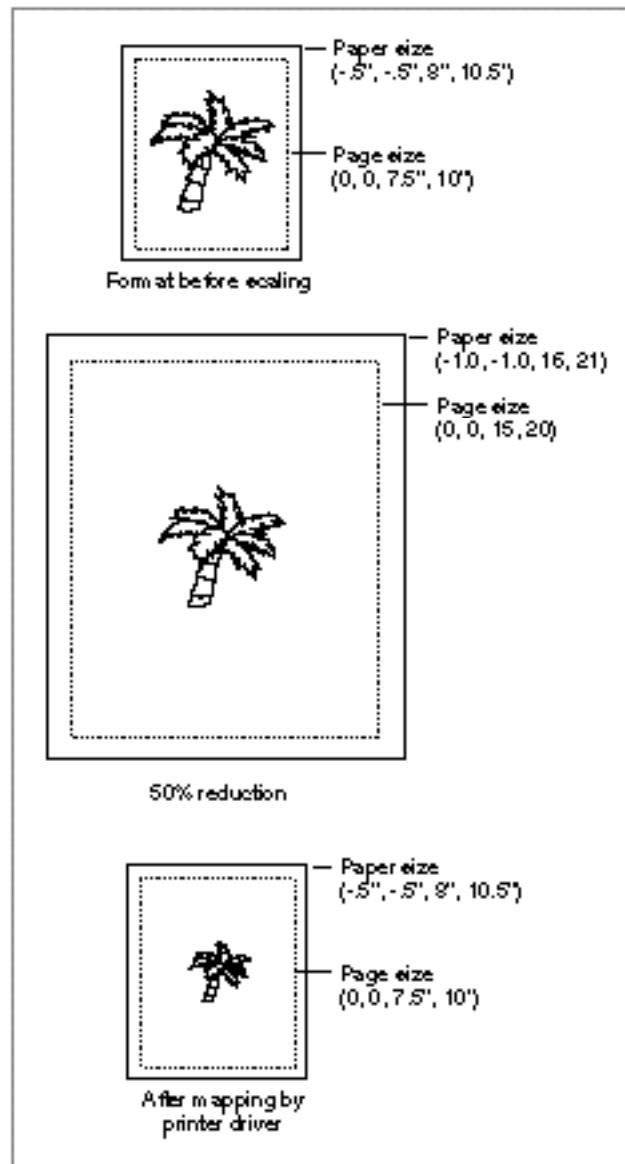
QuickDraw GX uses this mapping to scale page information into device pixels. A device pixel is the smallest physical area that a printer is capable of rendering. Typically, the mapping consists of the high-resolution scaling information needed to print a page at the highest quality.

QuickDraw GX and the printer driver set up the mapping. Your application can retrieve the mapping but cannot set it directly. You might want to retrieve it, for example, to set the mapping property of a view port to represent the printer on screen. For more information about view port objects, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.



Figure 3-6 shows how the scaling item affects the mapping.

**Figure 3-6**      Scaling a format object



When 50% scaling is applied, the scaling variables in the mapping are actually doubled, which causes the shape to appear the same size on a page of paper that is twice its original size. When the printer driver maps the page to dots-per-inch, it reduces the format dimension and everything within them, including the shape object. The result is that the shape is scaled to 50% when it is printed.

## Forms and Format Objects

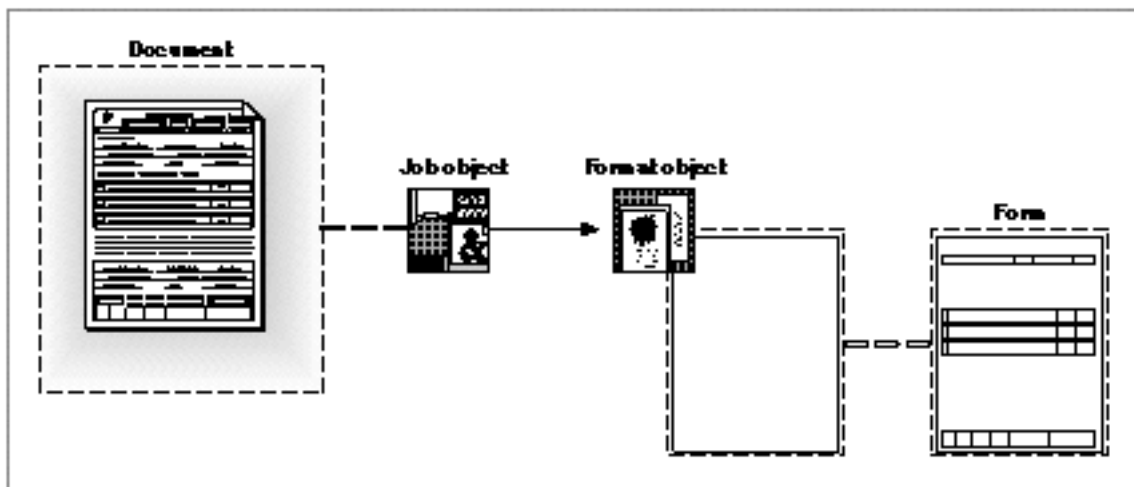
QuickDraw GX provides the form property, which allows your application to format pages of a document with a template. A form is made up of two shape objects—one shape defines the form itself, and the other shape, the mask shape, defines erasable areas within the form. The mask shape is optional; your application can erase the contents within a form, but this technique is not recommended.

Your application can specify a form for any format object associated with the formatting printer. Your application uses this form as a backdrop that is applied to a set of pages. For example, you can use a form to define erasable areas within pages created using a database application.

To associate a form shape and a mask shape with a format object associated with a page, you use the `GXSetFormatForm` function. To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. The shape type that you associate with a format object must be a picture shape.

Figure 3-7 shows a page from a document created by a database application. The figure also shows the job object corresponding to the document, the job's format object, and a form.

**Figure 3-7** Using a form to format a page



Forms save time during spooling, rendering, and I/O. During spooling, QuickDraw GX spools a form shape only once. QuickDraw GX renders a form shape once for each distinct format object it is attached to. During I/O, if the printer can cache the representation of the form, QuickDraw GX saves data transmission time by sending the form to the printer only when it has to.

## Halftones and Format Collections

---

You can use a halftone to represent more colors than can be represented on a printer by alternating available colors of a fixed cell size so that a noncontinuous tone device appears to produce continuous-tone grayscale or full-color images.

You can use the format collection to specify halftone information on a page-by-page basis. Initially, the printer driver specifies the halftone information for the default format by storing the information in this format's format collection object. You can add halftones to this collection, in which case you are changing the halftone for the entire document. You can also change the halftone information in a format collection associated with the format object for specific pages, in which case only the pages associated with the format object receive the halftone. Storing halftone information in a format collection is discussed in "Storing Halftone Information in a Format Collection," which begins on page 3-52.

### Note

To specify halftone information on a shape-by-shape basis, you use a synonym attached to the shape's ink object. For more information about the halftone synonym, see the chapter "Advanced Printing Features" in this book. <sup>u</sup>

The format-halftone item in the format collection specifies the halftones to use. The collection item specifies a `gxFormatHalftoneInfo` structure that defines the number of allowable halftones and their characteristics. For the definition of the `gxFormatHalftoneInfo` structure, see "Format-Halftone Information" on page 3-92.

The definition of each halftone is specified in a `gxHalftone` structure, which is described completely in the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*:

```
struct gxHalftone{
    fixed          angle;           /* direction of halftone */
    fixed          frequency;       /* cells per inch */
    gxDotType      method;          /* kind of pattern */
    gxTintType     tinting;         /* tint calculation method */
    gxColor        dotColor;        /* color of foreground */
    gxColor        backgroundColor; /* color of background */
    gxColorSpace   tintSpace;       /* color space for tint */
};
```

You can specify any number of these `gxHalftone` structures in the format-halftone information item. QuickDraw GX selects appropriate halftones from the list of available halftones in the item. Its selection is based upon the `tinting` field in the halftone structure:

- <sup>n</sup> When you print to a black-and-white PostScript device, QuickDraw GX looks for a halftone structure that specifies `gxLuminanceTint` in the `tinting` field. If no halftone specifies this value, it looks for a halftone that specifies `gxComponent4Tint` as its tinting method. Component 4 is the black component in the CMYK (cyan,

magenta, yellow, and black) space. If no halftone specifies this tinting method either, the first halftone in the list is used.

- n When you print to a color PostScript device, a maximum of four halftones are used. QuickDraw GX attempts to locate halftones for the following tint calculation methods: `gxComponent1Tint` for the cyan halftone, `gxComponent2Tint` for the magenta halftone, `gxComponent3Tint` field for the yellow halftone, and `gxComponent4Tint` for the black halftone. If a tinting method is in the list more than once, the first one in the list is used.

If a halftone for the `gxComponent4Tint` method is not in the list, QuickDraw GX uses the `gxLuminanceTint` tinting method for the black halftone. If the `gxLuminanceTint` tinting method cannot be found either, QuickDraw GX uses the first halftone in the list for the black halftone.

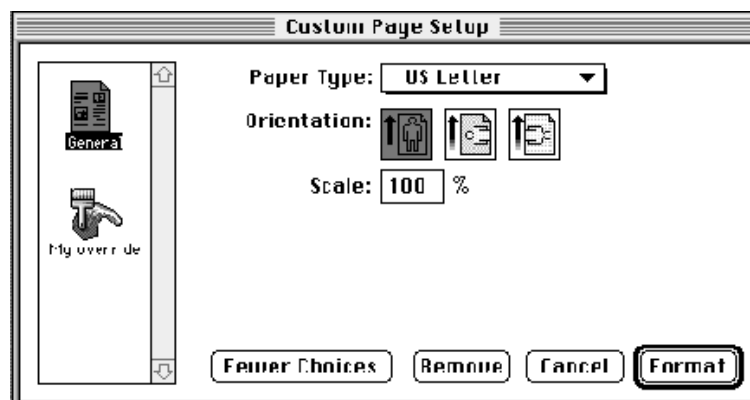
If QuickDraw GX cannot find a halftone for the `gxComponent1Tint`, `gxComponent2Tint`, or `gxComponent3Tint` tinting methods, it uses the black halftone for the missing tinting method.

It is only possible to use halftones to the extent that a particular PostScript device supports them. The dot color and background color of a halftone are ignored because QuickDraw GX assumes that the dot color for a black-and-white device is black and the dot color for a color device with the `gxComponent2Tint` tinting method is magenta.

## Dialog Box Customization

QuickDraw GX allows your application to customize print dialog boxes, typically, by adding panels. A panel is a portion of an expanded dialog box that presents additional printing options for users. For example, you may allow the user to specify custom margins in a panel you add to the Page Setup dialog box or the Custom Page Setup dialog box. Figure 3-8 shows the expanded Custom Page Setup dialog box with two panels, the General panel and the “My override” panel. The contents of the General panel are shown in Figure 3-8.

**Figure 3-8** The expanded Custom Page Setup dialog box with two panels



QuickDraw GX stores a user's responses to most default dialog items in collection objects. Your application can use collection objects to store information from panels you have added to dialog boxes. For information about storing items in collection objects and retrieving them, see "Using Printing-Related Collection Objects" beginning on page 3-27.

#### Note

If several applications want to provide the same option in the same panel, it may be better to implement the panel in a printing extension. For more information about printing extensions, see the printing extensions chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*. u

To create a panel, you must define a panel resource (`gxPrintPanelType`), as described in the next section. You may also define an extended item list resource (`gxExtendedDITLType`) that defines how to respond to user actions, such as clicking a button, while the panel is on the screen. This resource is described in the section "Automating Panel Events" beginning on page 3-25.

Messages are used to notify the application when a print dialog box is about to be displayed. This allows you to load the panel from the resource before the dialog box is displayed. The functions that invoke these messages are shown in Table 3-1.

**Table 3-1** Functions that enable dialog box panels

Function	Description
<code>GXJobPrintDialog</code>	Displays the Print dialog box
<code>GXJobDefaultFormatDialog</code>	Displays the Page Setup dialog box
<code>GXFormatDialog</code>	Displays the Custom Page Setup dialog box

Your application typically takes these steps to enable a panel when the user chooses a menu item that brings up a dialog box:

1. Call the appropriate function, such as `GXJobPrintDialog` if the user chose the Print menu item from the File menu.
2. Respond to the message, such as `gxJobPrintDialog`, by invoking your override function; for example, `MyJobPrintDialog`. This response was set up by the call to `GXInstallApplicationOverride` to set up the application as a message handler.
3. Set up the panel and call `GXSetupDialogPanel` to display it. These actions are performed by the override function.
4. Forward the message. This action is also performed by the override function.

For an overview of how messages allow you to display a panel in a print dialog box, see the chapter "Introduction to Printing With QuickDraw GX" in this book.

To forward a message, you call one of the functions in Table 3-2.

**Table 3-2** Functions that forward a dialog box message

Function	Description
Forward_GXJobPrintDialog	Forwards the <code>gxJobPrintDialog</code> message
Forward_GXJobDefaultFormatDialog	Forwards the <code>gxJobDefaultFormatDialog</code> message
Forward_GXFormatDialog	Forwards the <code>gxFormatDialog</code> message

You pass exactly the same arguments to the forwarding function as those with which your override function was called. For an example of setting up a custom dialog box with an added panel and forwarding a message, see the section “Adding Panels to Dialog Boxes” beginning on page 3-67. For information about how to forward other messages, see the Message Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## The Dialog Box Panel Resource

A panel resource defines a panel. It specifies the following information:

- n The panel’s name, such as `MyOverride`. The name appears in the list of panels for an extended dialog box, under the panel’s icon.
- n The script used to display the panel, such as `smRoman`.
- n The resource of the 'DITL' resource that defines the items in the panel.
- n The resource ID of the icon to display in the extended dialog box.

Listing 3-1 shows the structure of a panel resource.

**Listing 3-1** A panel resource definition template

```
type gxPrintPanelType {
    pstring[31];    /* name */
    integer Script; /* international script */
    fill word;      /* long word reserved for future use */
    fill word;      /* long word reserved for future use */
    integer;        /* the icon id */
    integer;        /* the ditl id */
};
```

## Responding to Panel Events

---

QuickDraw GX handles events for its default panels automatically. If you change a default panel or you add another one, you may need to override the messages that QuickDraw GX sends in order to process the items that you added.

If an event occurs while a panel is displayed, QuickDraw GX sends a `gxFilterPanelEvent` message. If you want to filter the event, you can override this message by installing a handler for it and by specifying a function that matches the prototype defined for `GXFilterPanelEvent` on page 3-124.

If you do not need to filter the event, you may choose to handle the event in your code, you may automate the handling of the event, or you may do both. Events that you need to handle in some way include mouse clicks on radio buttons or checkboxes, choosing an item from a pop-up menu, and keystrokes in editable text fields.

To handle the event in your application code, you install an override for the `gxHandlePanelEvent` message. You can override this message by installing a handler for it and specifying a function that matches the prototype defined for `GXHandlePanelEvent` on page 3-123.

For information about messages and how to override them, see the chapter “Introduction to Printing With QuickDraw GX” in this book. For an example of installing an override function, see the chapter “Core Printing Features” in this book. For information about automatically handling panel events, see the next section.

## Automating Panel Events

---

You can allow QuickDraw GX to automatically respond to selections in dialog box panels without you writing additional application code. QuickDraw GX provides an extended item list (`gxExtendedDITLType`) resource that loads values or settings of items and responds to changes to items in an item list (‘DITL’) resource. The item types for which QuickDraw GX can load values or settings and respond to changes in them include

- n radio buttons
- n checkboxes
- n pop-up menus
- n editable text in strings; the strings may represent characters, integers, and real numbers

QuickDraw GX obtains the values or settings from items in the job and format collections and responds to changes, by updating the information in these items, when the changes are confirmed. If the panel is part of the Print dialog box, the collection item must be in the job collection. If the panel is part of the Page Setup or Custom Page Setup dialog box, the collection item must be in the format collection.

For example, as a panel is displayed, an extended item resource specifies the collection item to use to set a group of radio buttons. If a user clicks on an unselected radio button, QuickDraw GX deselects the previously highlighted button and highlights the chosen

one. When the user closes the panel and confirms the settings (for example, by clicking the OK button), the items specified by the extended item resource are placed back in their collections. If the user cancels the panel, the collection items are not changed.

The processing for checkboxes is similar. If the checkbox is not checked, QuickDraw GX checks it; if it is checked, QuickDraw GX unchecks it. Editable text is checked when the panel is closed and confirmed.

QuickDraw GX uses the resource IDs of the extended item list resource and 'DITL' resources to determine which extended item list to associate with the item list. If both kinds of resources have the same ID, they are used together. Specifically, when an open-panel (`gxPanelOpenEvt`) message is sent in response to the user choosing a panel in a dialog box, QuickDraw GX uses the extended item list resource that corresponds to the panel's 'DITL' resource to load and set the items. (Recall from the previous section that the panel resource specifies a 'DITL' resource.)

Listing 3-2 shows the extended item resource definition for editable text that represents a real number.

**Listing 3-2** The extended item list resource definition template

```
#define  xdtlRadioButtons      0
#define  xdtlCheckBox         1
#define  xdtlEditTextInteger  2
#define  xdtlEditTextReal     3
#define  xdtlEditTextString   4
#define  xdtlPopUp            5

type gxExtendedDITLType {
...
    case EditTextReal:
        key      integer = xdtlEditTextReal;
        literal  longint; /* 4 byte id for storage in job
                           object or format object */
                           longint; /* numerical id for storage in
                           job object or format object */
        integer; /* offset in bytes into the item
        byte;    /* corresponding ditl item */
        byte;    /* 0 = dont select, 1 = select
        pstring[15];/* low bound - nil means 'I
                           don't care' */
        pstring[15];/* high bound - nil means 'I
                           don't care' */
...
};
};
```



There are common fields for each item type supported by an extended item list resource:

- n The tag ID, such as `gxPrintingTagID` for QuickDraw GX defined tags.
- n The ID of the collection item, such as `gxFormatHalftoneTag`.
- n The offset into the collection item. The offset allows you to specify several values, such as the settings for checkboxes, in the same item.
- n The corresponding item in the 'DITL' resource, starting with 1.

The remaining fields depend on the kind of data. For real number editable text, the fields specify the following:

- n Whether or not to highlight the field's contents when the panel is displayed; 0 specifies do not highlight, 1 specifies to highlight.
- n The lowest possible value for range checking. A `nil` string specifies no lower bound.
- n The highest possible value for range checking. A `nil` string specifies no upper bound.

If a user enters data that does not conform to the specified format or specifies a number that is out of range, the text item inverts, and a system beep alerts the user to the problem when the user attempts to leave the field.

For the definitions of other kinds of fields, see "The Extended Item List Resource" on page 3-128. For an example of specifying an extended item list resource, see "Setting Up Dialog Box Resources" on page 3-70.

## Using Printing-Related Collection Objects

---

Your application can use collection objects to store information associated with a particular document. To access collection objects used by QuickDraw GX printing features, you use functions provided by QuickDraw GX. You manipulate the pieces of information in collection objects using Collection Manager functions. The Collection Manager is described in *Inside Macintosh: QuickDraw GX Environment and Utilities*.

QuickDraw GX allows you to access a collection object associated with a job object, format object, or paper-type object. If you want to store or access printing-related information associated with a document in the job collection, you use the `GXGetJobCollection` function to access this collection object. If you want to store or access formatting information in the format collection, you use the `GXGetFormatCollection` function. If you want to store or access paper-type information in the paper-type collection, you use the `GXGetPaperTypeCollection` function.

You then specify the collection item in a call to `GetCollectionItem` to retrieve the specific data from a collection. The collection item corresponds to the data you wish to retrieve. For example, the `gxCopiesTag` collection item specifies access to data in the `gxCopiesInfo` data structure:

```
enum {gxCopiesTag = 'copy'};

struct gxCopiesInfo{
    long copies;
};
```

#### Note

The collection tags, collection item tags, and structures for collection objects are defined in the section “Constants and Data Types” beginning on page 3-133. For complete information about using collections, see the Collection Manager chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. u

## Accessing Data From a Collection Object

---

The following example shows you how to access an item from a collection object. For an example of adding an item to a collection object, see “Associating Format Objects With Document Pages” on page 3-61. For an example of replacing a collection item, see “Using a Collection to Implement the Print One Copy Menu Item” on page 3-29.

Listing 3-3 shows how to use the `GXGetJobCollection` function to access the number of copies, which is stored in a job collection.

---

### Listing 3-3      Accessing copies information stored in a job collection

```
OSErr MyGetJobCopies(MyDocumentPtr myDocument, long *numCopies)
{
    OSErr          err;
    Collection      jobCollection;
    gxCopiesInfo    theCopiesInfo;
    long            dataSize = sizeof(theCopiesInfo);

    /*
       Get the job collection and look for a gxCopiesTag collection
       object item.
    */
    jobCollection = GXGetJobCollection(myDocument->documentJob);
    err = GetCollectionItem(jobCollection,
                           gxCopiesTag,
```

```

                                gxPrintingTagID,
                                dataSize,
                                &theCopiesInfo);

    /* Extract the number of copies and return it. */
    if (err == noErr)
        *numCopies = theCopiesInfo.copies;
    return err;
}

```

## Using a Collection to Implement the Print One Copy Menu Item

---

To implement the Print One Copy menu item, you must change items in the job collection. You must ensure that only one copy will be printed, that all pages will be printed, and that the output will actually go to a printer and not to a print file. After the copy has been printed, you must set the contents of the collection items back to their original values so that the user's settings are preserved.

Listing 3-4 shows how to set the values of the `gxCopiesTag`, `gxPageRangeInfo`, and `gxFileDestinationTag` items so that only one copy of all pages is printed and it is sent to the printer. It also shows how to restore the original values for these collection items after the print operation has been completed.

---

**Listing 3-4**      Modifying the job collection to implement the Print One Copy menu item

```

OSErr MyPrintOneCopy(MyDocumentPtr whichDocument)
{
    OSErr                err;
    Collection            jobCollection;
    gxCopiesInfo          copiesInfo;
    gxFileDestinationInfo destInfo;
    gxPageRangeInfo       pageRangeInfo;
    Ptr                  oldCopiesInfo = nil;
    Ptr                  oldPageRangeInfo = nil;
    Ptr                  oldDestInfo = nil;
    long                 oldCopiesSize;
    long                 oldPageRangeInfoSize;
    long                 oldDestInfoSize;

    /* Get the job collection and set it up to print one copy */
    jobCollection = GXGetJobCollection(whichDocument->documentJob);

```

## Page Formatting and Dialog Box Customization

```

/* Set number of copies to 1 */
copiesInfo.copies = 1;
err = MyReplaceCollectionItem(&copiesInfo,
                             sizeof(gxCopiesInfo),
                             gxCopiesTag, gxPrintingTagID,
                             jobCollection, &oldCopiesInfo,
                             &oldCopiesSize);
nrequire(err, ReplaceCopies_error);

/* Set page range to "all". */
pageRangeInfo.simpleRange.optionChosen = gxDefaultPageRange;
pageRangeInfo.minFromPage = 1;
pageRangeInfo.simpleRange.fromPage = 1;
pageRangeInfo.maxToPage = whichDocument->numPages;
pageRangeInfo.simpleRange.toPage = whichDocument->numPages;
pageRangeInfo.simpleRange.printAll = true;
err = MyReplaceCollectionItem(&pageRangeInfo,
                             sizeof(gxPageRangeInfo),
                             gxPageRangeTag, gxPrintingTagID,
                             jobCollection, &oldPageRangeInfo,
                             &oldPageRangeInfoSize);
nrequire(err, ReplacePageRange_error);

/* Set destination to "printer". */
destInfo.toFile = false;
err = MyReplaceCollectionItem(&destInfo,
                             sizeof(gxFileDestinationInfo),
                             gxFileDestinationTag, gxPrintingTagID,
                             jobCollection, &oldDestInfo,
                             &oldDestInfoSize);
nrequire(err, ReplaceDestination_error);

/* Print one copy of the document. (not shown here) */
err = MyPrintDocument(whichDocument);

/*
   Restore original number of copies, page range, and output
   destination in case it needs to be reused.
*/
ReplaceCopies_error:
    MyReplaceCollectionItem(oldCopiesInfo, oldCopiesSize,
                           gxCopiesTag, gxPrintingTagID,
                           jobCollection, nil, nil);

```

## Page Formatting and Dialog Box Customization

```

ReplacePageRange_error:
    MyReplaceCollectionItem(oldPageRangeInfo, oldPageRangeInfoSize,
                            gxPageRangeTag, gxPrintingTagID,
                            jobCollection, nil, nil);
ReplaceDestination_error:
    MyReplaceCollectionItem(oldDestInfo, oldDestInfoSize,
                            gxFileDestinationTag, gxPrintingTagID,
                            jobCollection, nil, nil);

    /* Dispose of pointers created by MyReplaceCollectionItem */
    if (oldCopiesInfo)
        DisposePtr(oldCopiesInfo);
    if (oldPageRangeInfo)
        DisposePtr(oldPageRangeInfo);
    if (oldDestInfo)
        DisposePtr(oldDestInfo);

    return err;
}

```

## Replacing Items in Collections

---

The `MyReplaceCollectionItem` function is a generic routine that you could write to replace collection items. In the implementation in Listing 3-5, the data being replaced is returned in a variable pointed to by `oldData`, unless the pointer is `nil`. If the item does not exist, the new data is returned via the pointer instead.

---

**Listing 3-5**     Replacing collection items

```

OSErr MyReplaceCollectionItem(void *newData, long collectSize,
                              OSType collectType, long collectID,
                              Collection whichCollection,
                              Ptr *oldData, long *oldDataSize)
{
    OSErr    err;
    long     index;

    /*
     * If returning the old data, get it.
     * If there is no old data, return a copy of the new data.
     */
    if (oldData)

```

```

{
    err = GetCollectionItemInfo(whichCollection,
                                collectType,
                                collectID,
                                dontWantIndex,
                                oldDataSize,
                                dontWantAttributes);

    if (err)
    {
        *oldDataSize = collectSize;
        *oldData = NewPtrSys(*oldDataSize);
        if (!(err = MemError()))
            BlockMove(newData, *oldData, collectSize);
    }
    else
    {
        *oldData = NewPtrSys(*oldDataSize);
        if (!(err = MemError()))
            err = GetCollectionItem(whichCollection,
                                    collectType,
                                    collectID,
                                    dontWantSize,
                                    *oldData);
    }
    nrequire(err, CouldNotSetOldData);
}

/*
    Add a new collection item; otherwise, get the existing
    item's index and replace the old collection item.
*/
err = AddCollectionItem(whichCollection,
                        collectType,
                        collectID,
                        collectSize,
                        newData);

```

```

if (err == collectionItemLockedErr)
{
    err = GetCollectionItemInfo(whichCollection,
                                collectType,
                                collectID,
                                &index,
                                dontWantSize,
                                dontWantAttributes);

    if (!err)
        err = ReplaceIndexedCollectionItem(whichCollection,
                                            index,
                                            collectSize,
                                            newData);
}

CouldNotSetOldData:
    return err;
}

```

## Specifying Page Ranges in the Job Collection

---

You can specify the page range and page range constraints in the page-range information job collection item. QuickDraw GX provides three kinds of representations for page ranges: simple numeric From and To values called the default page range, a single editable text field that specifies a replacement page range, and alphanumeric From and To values called a customized page range.

Listing 3-4 on page 3-29 shows how to set up a default page range for all pages to support the Print One Copy menu item of the File menu. The examples that follow show how to set up default, replacement, and customized page information for specific pages.

Listing 3-6 on page 3-33 shows how to set up the default page range for pages 1 through 4. The user may change these values to any within 1 and 9999.

---

### Listing 3-6      Setting up a default page range

```

OSErr MyConfigurePageRange1(MyDocumentPtr myDocument)
{
    OSERR          err;
    gxPageRangeInfo **pageRangeHdl;

    /*
        Create a handle to store the page range collection item in,
        and then retrieve the collection item.
    */
}

```

## Page Formatting and Dialog Box Customization

```

pageRangeHdl
    = (gxPageRangeInfo **)NewHandleClear(sizeof(gxPageRangeInfo));
nrequire_action(err, NewHandleClear_Failed, err = MemError());

err = GetCollectionItemHdl
    (GXGetJobCollection(myDocument->documentJob),
     gxPageRangeTag,
     gxPrintingTagID,
     (Handle) pageRangeHdl);
nrequire(err, GetCollectionItemHdl_Failed);

/*
    Use the standard "From-To" editable text field containing
    default numeric values.
    Specify that the "all pages" radio button is not to be
    selected and that the "From" field contains 1 and the
    "To" field contains 4.
*/
(*pageRangeHdl)->simpleRange.optionChosen = gxDefaultPageRange;
(*pageRangeHdl)->simpleRange.printAll = false;

(*pageRangeHdl)->simpleRange.fromPage = 1;
(*pageRangeHdl)->simpleRange.toPage = 4;
(*pageRangeHdl)->minFromPage = 1;
(*pageRangeHdl)->maxToPage = 9999;

/* Add (or replace) the collection item, and dispose of its
   handle. */
err = AddCollectionItemHdl(
    GXGetJobCollection(myDocument->documentJob),
    gxPageRangeTag,
    gxPrintingTagID,
    (Handle) pageRangeHdl);

GetCollectionItemHdl_Failed:
    DisposHandle((Handle) pageRangeHdl);

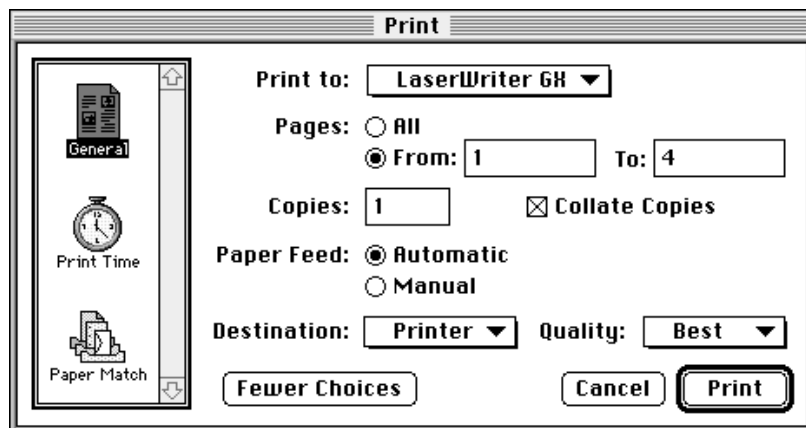
NewHandleClear_Failed:
    return err;
}

```



Figure 3-9 shows the Print dialog box after executing the code to set the page range. QuickDraw GX obtains the page range to display from the collection item.

**Figure 3-9** Print dialog box with default page range



Listing 3-7 shows how to set up a replacement page range, in which the From and To fields are replaced by a single editable text field. Note that the default editable text field is only one character, therefore, you almost always increase the size of the handle to accommodate the page range. The page range is a Pascal-style string.

**Listing 3-7** Setting up a replacement page range

```
OSErr MyConfigurePageRange2(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxPageRangeInfo **pageRangeHdl;

    /*
       Create a handle to store the page range collection item in,
       and then retrieve the collection item.
    */
    pageRangeHdl
        = (gxPageRangeInfo **)NewHandleClear(sizeof(gxPageRangeInfo));
    nrequire_action(err, NewHandleClear_Failed, err = MemError());;
```

## Page Formatting and Dialog Box Customization

```

err = GetCollectionItemHdl
    (GXGetJobCollection(myDocument->documentJob),
     gxPageRangeTag,
     gxPrintingTagID,
     (Handle) pageRangeHdl);
nrequire(err, GetCollectionItemHdl_Failed);

/*
    Replace the standard "From-To" editable text fields, with a
    single editable text field that contains "Chapter 5."
    Specify that the "all pages" radio button is not to be
    selected.
*/
(*pageRangeHdl)->simpleRange.optionChosen = gxReplacePageRange;
(*pageRangeHdl)->simpleRange.printAll = false;
SetHandleSize((Handle) pageRangeHdl,
    sizeof(gxPageRangeInfo) +titleSize-1);
nrequire_action(err, SetHandleSize_Failed, err = MemError());
BlockMove(FromToTitle, (*pageRangeHdl)->replaceString,
    titleSize);

/* Add (or replace) the collection item, and dispose of its
   handle. */
err = AddCollectionItemHdl(
    GXGetJobCollection(myDocument->documentJob),
    gxPageRangeTag,
    gxPrintingTagID,
    (Handle) pageRangeHdl);

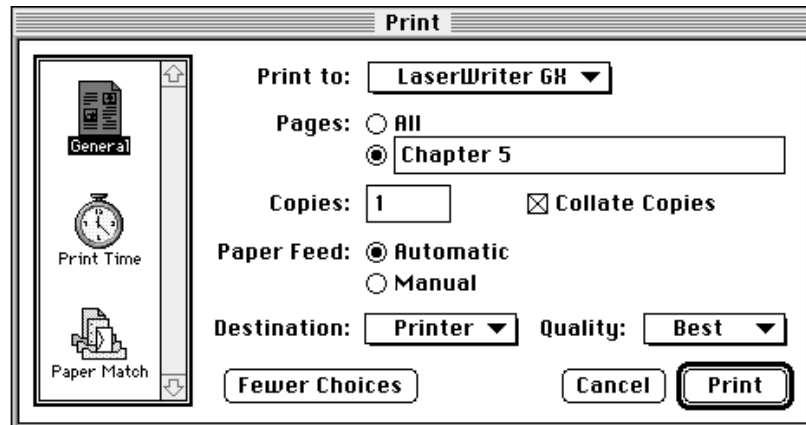
SetHandleSize_Failed:
GetCollectionItemHdl_Failed:
    DisposHandle((Handle) pageRangeHdl);

NewHandleClear_Failed:
    return err;
}

```

Figure 3-10 shows the Print dialog box after executing the replacement page range code. The contents of the title, “Chapter 5,” are displayed in a single editable text field. You must check for the validity of this field if the user changes it. For more information about parsing a page range to test its validity, see “Parsing Page Ranges” on page 3-73.

**Figure 3-10** Print dialog box with replacement page range



Listing 3-8 shows how to set up a customized page range, in which the From and To fields allow editable text.

**Listing 3-8** Setting up a customized page range

```
OSErr MyConfigurePageRange3(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxPageRangeInfo **pageRangeHdl;

    /*
        Create a handle to store the page range collection item in,
        and then retrieve the collection item.
    */
    pageRangeHdl
        = (gxPageRangeInfo **)NewHandleClear(sizeof(gxPageRangeInfo));
    nrequire_action(err, NewHandleClear_Failed, err = MemError());
```

## Page Formatting and Dialog Box Customization

```

err = GetCollectionItemHdl
    (GXGetJobCollection(myDocument->documentJob),
     gxPageRangeTag,
     gxPrintingTagID,
     (Handle) pageRangeHdl);
nrequire(err, GetCollectionItemHdl_Failed);

/*
    Use the standard "From-To" editable text fields, but they
    now contain a custom format for the page range values.
    Specify that the "all pages" radio button is not to be
    selected and that the "From" field contains "iii" and the
    "To" field contains "VI".
*/
(*pageRangeHdl)->simpleRange.optionChosen =
    gxCustomizePageRange;
(*pageRangeHdl)->simpleRange.printAll = false;
BlockMove("iii", &(*pageRangeHdl)->fromString[1], 3);
(*pageRangeHdl)->fromString[0] = 3;
BlockMove("VI", &(*pageRangeHdl)->toString[1], 2);
(*pageRangeHdl)->toString[0] = 2;

/* Add (or replace) the collection item, and dispose of its
   handle. */
err = AddCollectionItemHdl(
    GXGetJobCollection(myDocument->documentJob),
    gxPageRangeTag,
    gxPrintingTagID,
    (Handle) pageRangeHdl);

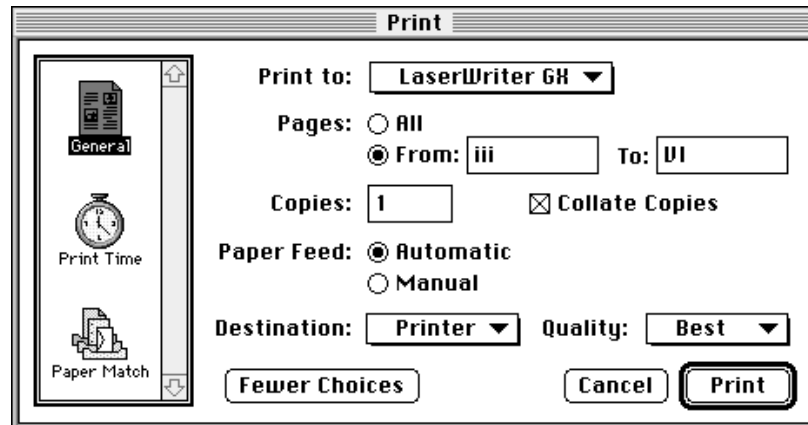
GetCollectionItemHdl_Failed:
    DisposeHandle((Handle) pageRangeHdl);

NewHandleClear_Failed:
    return err;
}

```

Figure 3-11 shows the Print dialog box after executing the customized page range code. The contents of the From and To fields are now editable text. You must check for the validity of these fields if the user changes them. For more information about parsing a page range to test its validity, see “Parsing Page Ranges” on page 3-73.

**Figure 3-11** Print dialog box with customized page range



## Using Format Objects and Collection Items to Format Pages

To support page-formatting features, your application needs to manipulate format objects and keep track of the number of times a format object is shared. Generally, you work with format objects when a user creates a new format, wants to share a format with additional pages in a single document, disposes of a page, or modifies a format that is shared by other pages in the same document. Because your application is responsible for associating format objects with the individual pages of a document, you need to save this association when a user saves a document.

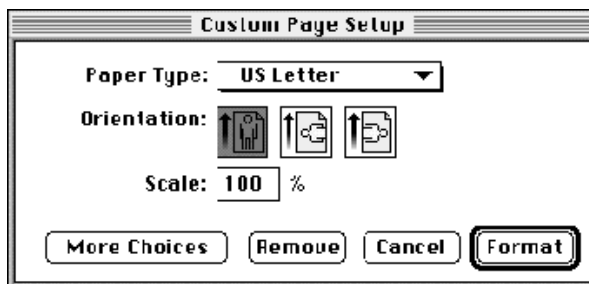
Your application can also manipulate format objects to support special formatting features. These features include associating form shapes with format objects, supporting page-specific halftone information in your application's documents, and copying format objects for use in multiple documents. QuickDraw GX also allows you to access information associated with a specific format object, such as its mapping and associated paper-type objects.

The following sections describe how to use page-formatting features.

## Creating a Format Object for a Page in a Document

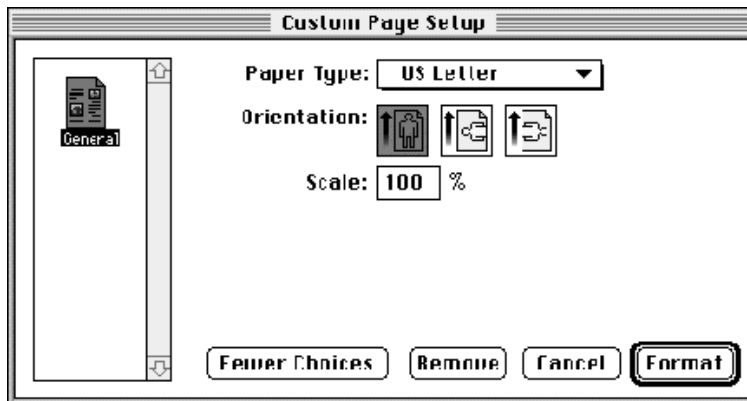
When a user wants to create a unique format or change its settings, the user chooses the Custom Page Setup menu item from the File menu. In response, you need to call the `GXFormatDialog` function to display the Custom Page Setup dialog box on the user's screen. Figure 3-12 shows the Custom Page Setup dialog box.

**Figure 3-12** The Custom Page Setup dialog box



If the user clicks the More Choices button in the Custom Page Setup dialog box, QuickDraw GX expands the dialog box. Figure 3-13 shows the expanded Custom Page Setup dialog box.

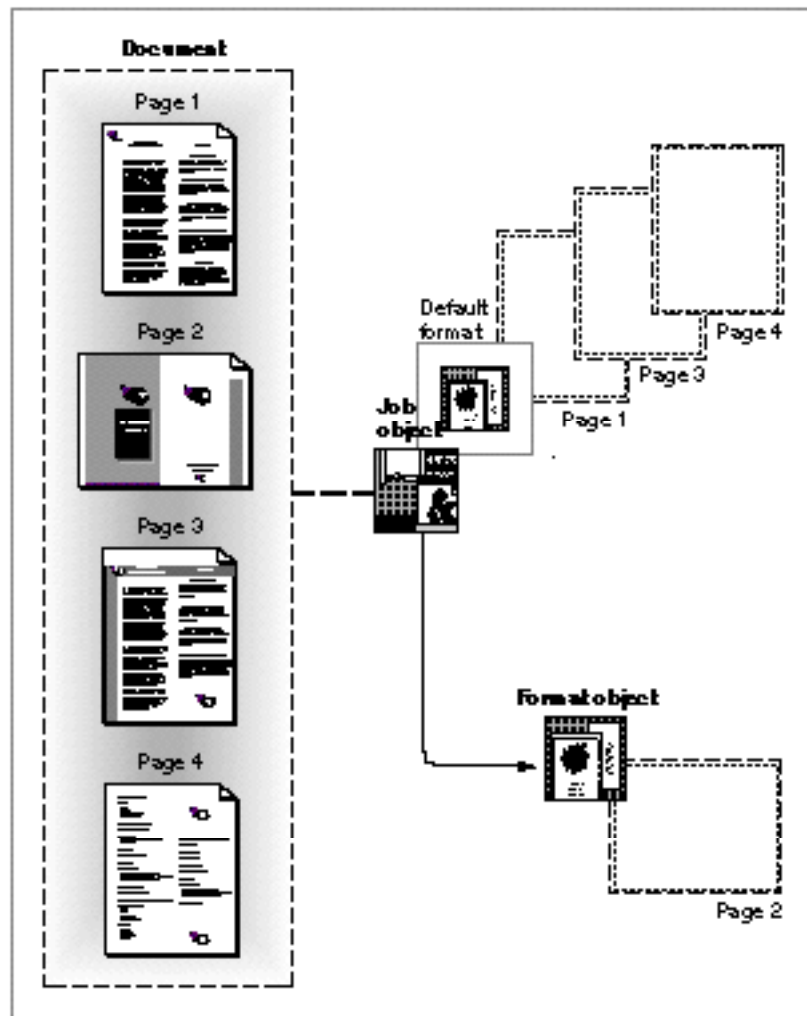
**Figure 3-13** The expanded Custom Page Setup dialog box



You need to create a new format object when a user chooses the Format button in the Custom Page Setup dialog box. For example, a user may create a four-page document, move to page 2, and then choose landscape orientation in the Custom Page Setup dialog box. The change to the page occurs when the user chooses the Format button.

Figure 3-14 shows a four-page document in which the second page uses a new format. Pages 1, 3, and 4 use the default format. Pages 1, 3, and 4 use the default format.

**Figure 3-14** A four-page document in which page two uses a unique format object



Listing 3-9 shows how to create a new format object for a single page in a document. You should note that the `GXNewFormat` function sets the owner count for the new format object to 1.

---

**Listing 3-9**      Creating a format object for a page in a document

```
OSErr MyPageFormatDialog(MyDocumentPtr myDocument)
{
    OSErr          err = noErr;
    gxDialogResult  result;
    gxEditMenuRecord editMenuRec;
    gxFormat        pageFormat;
    Boolean         newPgFormat = false;

    /* Fill in the location of your application's Edit menu items. */
    editMenuRec.editMenuID = mEdit;
    editMenuRec.cutItem     = kCut;
    editMenuRec.copyItem    = kCopy;
    editMenuRec.pasteItem   = kPaste;
    editMenuRec.clearItem   = kClear;
    editMenuRec.undoItem    = kUndo;

    /* Modify existing format object, else create a new one. */
    if (myDocument->pageFormat[myDocument->curPage - 1] != nil)
        pageFormat = myDocument->pageFormat[myDocument->curPage - 1];
    else
    {
        pageFormat = GXNewFormat(myDocument->documentJob);
        newPgFormat = true;
        err = GXGetJobError(myDocument->documentJob);
    }

    /* If no errors, display the Page Setup dialog box. */
    if (err == noErr)
    {
        result = GXFormatDialog(pageFormat, &editMenuRec, nil);
    }
}
```



## Page Formatting and Dialog Box Customization

```

    /*
       If the user chooses Remove, use the default format for
       this page. If the user chooses Format, store the new
       format with this page. If the user chooses Cancel,
       dispose of the cloned copy of the default format.
    */
    */
    switch (result)
    {
        case gxRevertSelected:
            GXDisposeFormat(pageFormat);
            pageFormat = nil;

        case gxOKSelected:
            myDocument->pageFormat[myDocument->curPage -1] =
                                                    pageFormat;

            /*
               Place code here if your application needs to
               adjust the document based on the new format object.
            */
            ...
            break;

        case gxCancelSelected:
            /*
               If the user selects Cancel, dispose of the cloned
               copy of the default format object.
            */
            if (newPgFormat) GXDisposeFormat(pageFormat);
            break;
    }
}
return err;
}

```

Within the Custom Page Setup dialog box used to create unique formats, the user has the option to return to the default format by choosing the Remove button or to not save the format by choosing the Cancel button. These options are handled by the `gxRevertSelected` and `gxCancelSelected` cases, respectively, of the switch statement in Listing 3-9.

For example, if the user decides that the second page in a document should not have a unique format, the user may choose the Remove button in the Custom Page Setup dialog box. When the user chooses this button, your application needs to dispose of the format object for this page and reassociate it with the default format.

Although a user may choose to create a new format for a page using the Custom Page Setup dialog box, the user may also decide not to save this format.

For example, a user may click on page 4 of a document, choose the Custom Page Setup dialog box, and modify the scaling of this page. The user may then decide not to save the new scaling information and choose the Cancel button in the dialog box. When the user chooses the Cancel button, your application needs to dispose of the newly created format object and reassociate this page with its previously saved format object.

Saving a job object and the format objects it references is discussed in the chapter “Core Printing Features” in this book.

## Sharing Formats for Document Pages

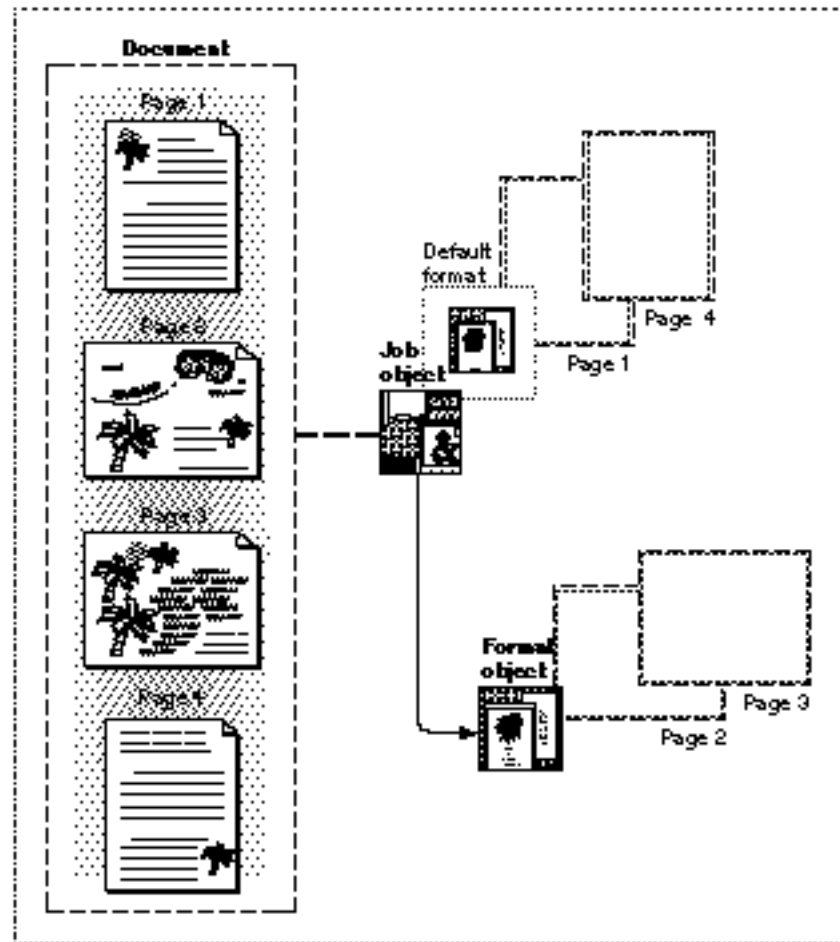
---

You need to clone a format object when a user wants to share a format, created using the Custom Page Setup dialog box, with an additional page in the same document. For example, a user may have a four-page document that consists of one page in landscape orientation and three pages that use the default format. A user may decide that page 3 of this document should also use landscape orientation.

When the user clicks on page 3 and chooses the Format button in the Custom Page Setup dialog box, you need to clone the format object currently used for page 2 in this document.

Figure 3-15 shows a four-page document in which the second and third pages use the same format. Pages 1 and 4 use the default format.

**Figure 3-15** A four-page document in which pages 2 and 3 use the same format object



Listing 3-10 shows how to clone a format object when it becomes shared. You should note that the `GXCloneFormat` function increments the owner count of this format object by 1. In this example, the format object is shared by two pages in a single document, so its owner count is also 2.

---

**Listing 3-10** Cloning a format object for two pages in a document

```
OSErr MyApplyPageFormat(MyDocumentPtr myDocument,
                        gxFormat aNewFormat)
{
    OSErr      err = noErr;
    gxFormat    pageFormat;

    /*
       If the specified format object is not the same as the
       default format, clone it so it can be shared by different
       pages. If it is the default format, set the reference to
       nil, which specifies using the default format.
    */
    if ((aNewFormat != nil) &&
        (aNewFormat != GXGetJobFormat(myDocument->documentJob, 1)))
    {
        pageFormat = GXCloneFormat(aNewFormat);
        err = GXGetJobError(myDocument->documentJob);
    }
    else
        pageFormat = nil;

    /*
       If there are no errors, dispose of the old format object and
       store the new one. Reformat the page, if necessary.
    */
    if (err == noErr)
    {
        if (myDocument->pageFormat[myDocument->curPage - 1] != nil)
            GXDisposeFormat
                (myDocument->pageFormat[myDocument->curPage - 1]);
        myDocument->pageFormat[myDocument->curPage - 1] = pageFormat;
    }
}
```

```
    /*  
        Place code here if your application needs to adjust the  
        document based on the new format object.  
    */  
    ...  
}  
return err;  
}
```

## Disposing of a Format Object for a Page in a Document

---

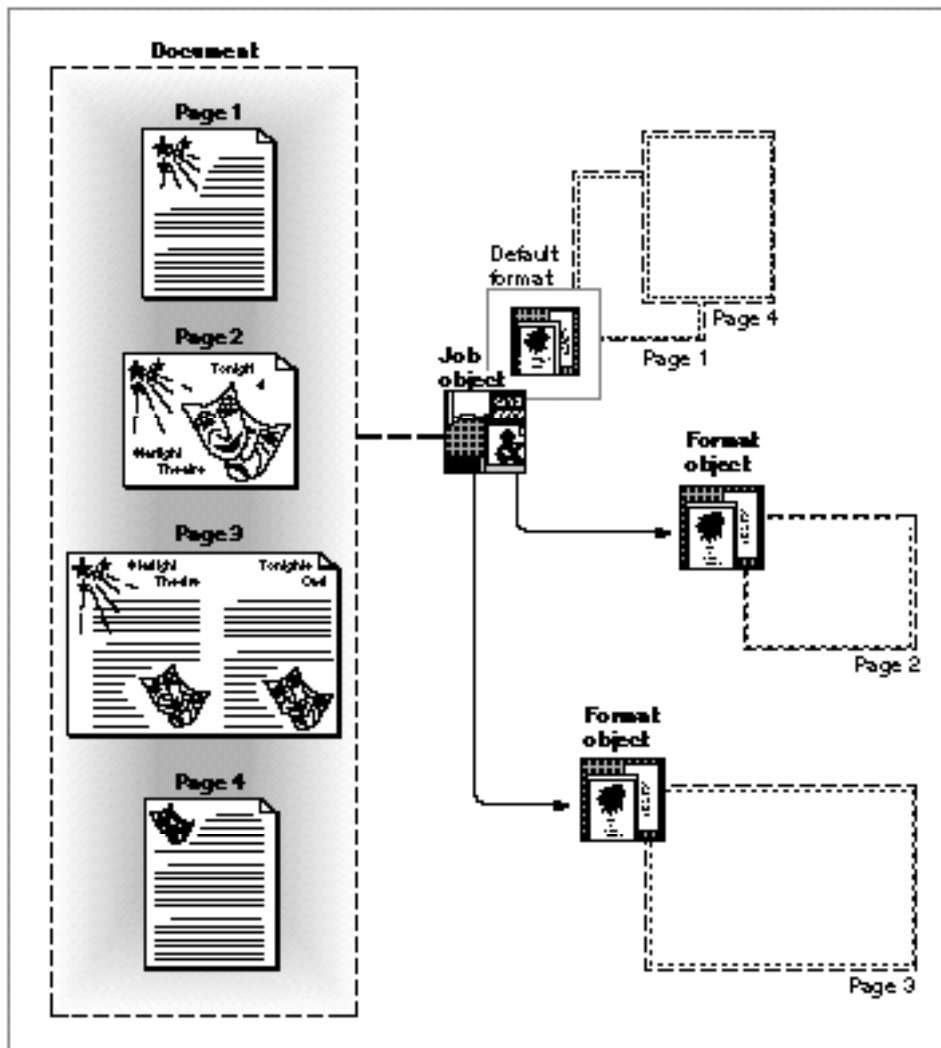
You need to dispose of a format object when a user wants to modify a format for a single page that is also shared by other pages in the same document, the user wants to return to the default format, or the user decides not to save a format.

For example, a user may have a four-page document that consists of two pages in landscape orientation (pages 2 and 3) and two pages that use the default format (pages 1 and 4). A user may decide to modify the scaling of page 3 of this document. A user specifies scaling for a page in the Custom Page Setup dialog box. Note that a user also can modify scaling for the default format in the Page Setup dialog box.

When the user clicks on page 3, specifies a scaling factor, and chooses the Format button in the Custom Page Setup dialog box, you need to dispose of the format object for this page and create a new one. This user is not modifying page 2, and therefore, you should not modify or dispose of its format object.

Figure 3-16 shows a four-page document in which the second and third pages use landscape orientation, but page 3 uses a modified scaling factor. Pages 1 and 4 use the default format.

**Figure 3-16** A four-page document in which pages 2 and 3 use unique formats objects



Listing 3-11 shows how to dispose of a format object for a page in a document. In this example, you need to dispose of the format object because it is shared by another page in the document (its owner count is greater than 1).

The `GXDisposeFormat` function decrements the owner count of this format object by 1. In this example, the format object is now used by only one page in the document, so its owner count becomes 1. Storage for a format object is removed only when its owner count becomes 0. After you call the `GXDisposeFormat` function, you need to call the `GXNewFormat` function to create a new format object for this page.

---

**Listing 3-11** Disposing of a format object for a page in a document and creating a new one

```
OSErr MyDeletePage(MyDocumentPtr myDocument)
{
    OSErr    err;
    long     curPage, pg;

    /*
     * Dispose of the current page's shape object and format
     * object.
     */
    curPage = myDocument->curPage;
    GXDisposeShape(myDocument->documentPage[curPage - 1]);
    if (myDocument->pageFormat[curPage - 1] != nil)
        GXDisposeFormat(myDocument->pageFormat[curPage - 1]);

    /* Place application-specific code to delete a page here. */
    ...

    /*
     * Shift all pages coming after this one to fill the gap
     * created by this deletion. When finished, decrement the
     * number of pages in the document.
     */
    if (myDocument->numPages != 0)
        for (pg = curPage; pg < myDocument->numPages; pg++)
        {
            myDocument->documentPage[pg - 1] =
                myDocument->documentPage[pg];
            myDocument->pageFormat[pg - 1] =
                myDocument->pageFormat[pg];
        }
    --myDocument->numPages;
```

## Page Formatting and Dialog Box Customization

```

/* If the current page is beyond the last page, reset it. */
if (curPage > myDocument->numPages)
    --myDocument->curPage;

/*
    Invalidate the window so that the page is updated on screen.
    Check for errors and return.
*/
InvalRect(&(myDocument->documentWindow)->portRect);
err = GXGetJobError(myDocument->documentJob);
if (err == noErr) err = (OSErr)GXGetGraphicsError(nil);
return err;
}

```

## Using Forms With Format Objects

---

Your application may choose to support a form that can be applied to each page in a document. This may save printing time because the form can be stored in the printer's memory and need not be sent with each page of the document. For an introduction to forms, see "Forms and Format Objects," which begins on page 3-20.

To associate a form shape and its mask shape with a format object, you use the `GXSetFormatForm` function. To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. The shape type that you associate with a format object must be a picture shape.

The `GXSetFormatForm` function replaces any form previously associated with a particular format object. It increments the owner counts of the new picture shapes (by calling the `GXCloneShape` function) and decrements the owner count of the old picture shapes (by calling the `GXDisposeShape` function).

Listing 3-12 shows how to associate a form with a format object. The `MyAddFormatForm` function in the listing adds a form consisting of a rectangle to the format object of the current page.



**Listing 3-12** Adding a form to a format object

---

```

OSErr MyAddFormatForm(MyDocumentPtr myDocument)
{
    OSERR          err;
    long           curPage;
    gxFormat       theFormat;
    gxShape        rectShape;
    gxRectangle    pageRect;

    /*
     * Get the current format object. If it's nil, use the job's
     * default format object.
     */
    curPage = myDocument->curPage;
    theFormat = myDocument->pageFormat[curPage - 1];

    if (theFormat == nil)
        theFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /*
     * Create a rectangle shape to use as the format object's form.
     * Make the rectangle's frame the imageable area of the page.
     */
    GXGetFormatDimensions(theFormat, &pageRect, nil);
    rectShape = GXNewRectangle(&pageRect);
    GXSetShapeBounds(rectShape, &pageRect);
    GXSetShapePen(rectShape, ff(3));
    GXSetShapeFill(rectShape, gxClosedFrameFill);
    err = (OSERR) GXGetGraphicsError(nil);

    /*
     * Set the format object's form to a new picture shape, check
     * for errors, and then dispose of the shape.
     */
    if (err == noErr)
    {
        GXSetShapeType(rectShape, gxPictureType);
        GXSetFormatForm(theFormat, rectShape, nil);
        err = GXGetJobError(myDocument->documentJob);
    }
    GXDisposeShape(rectShape);
    return err;
}

```

## Storing Halftone Information in a Format Collection

---

Your application can store halftone information for each page in a document in a format collection. QuickDraw GX stores the halftone structure for a format object as a collection item in the format collection. For an introduction to printing with halftones, see “Halftones and Format Collections,” which begins on page 3-21.

Halftones are described by the `gxHalftone` structure definition:

```
struct gxHalftone{
    fixed          angle;
    fixed          frequency;
    gxDotType      method;
    gxTintType      tinting;
    gxColor         dotColor;
    gxColor         backgroundColor;
    gxColorSpace    tintSpace;
};
```

The `angle` parameter describes the direction of the halftone. The `frequency` parameter describes the size of the dot, in cells per inch. The `method` parameter describes the way in which the halftone cell is filled. The `tinting` parameter describes how the desired color is converted into a ratio of color dots and background dots. The `dotColor` and `backgroundColor` parameters are the colors of the dots used to form the halftone. And the `tintSpace` parameter describes the color space that the original color is converted to before the tint value is determined. For detailed information on the `gxHalftone` structure, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

The `gxFormatHalftoneTag` enumerator is used to identify the `gxFormatHalftoneInfo` structure in the format collection:

```
enum { gxFormatHalftoneTag = 'half' };

struct gxFormatHalftoneInfo{
    long          numHalftones;
    gxHalftone    halftones[1];
};
```

The `numHalftones` field specifies how many `gxHalftone` entries are in the `gxFormatHalftoneInfo` structure. The `halftones` field specifies each of them.

Listing 3-13 shows how to store halftone information for a page in a format collection.

---

**Listing 3-13** Storing halftone information in a format collection

```

OSErr MySetFormatHalftones(gxFormat theFormat,
                           gxFormatHalftoneInfo *theFormatHalftones)
{
    OSERR          err;
    Collection      fmtCollection;

    /*
     * Get the format collection, and attempt to delete a
     * gxFormatHalftoneTag collection item, in case one exists.
     * Then, add a new one.
     */
    fmtCollection = GXGetFormatCollection(theFormat);
    RemoveCollectionItem(fmtCollection,
                        gxFormatHalftoneTag,
                        gxPrintingTagID);
    err = AddCollectionItem(fmtCollection,
                        gxFormatHalftoneTag,
                        gxPrintingTagID,
                        sizeof(gxFormatHalftoneInfo),
                        theFormatHalftones);

    /*
     * Since we changed the format object's collection items, we
     * must call GXChangedFormat.
     */
    if (err == noErr)
        GXChangedFormat(theFormat);
    return err;
}

```

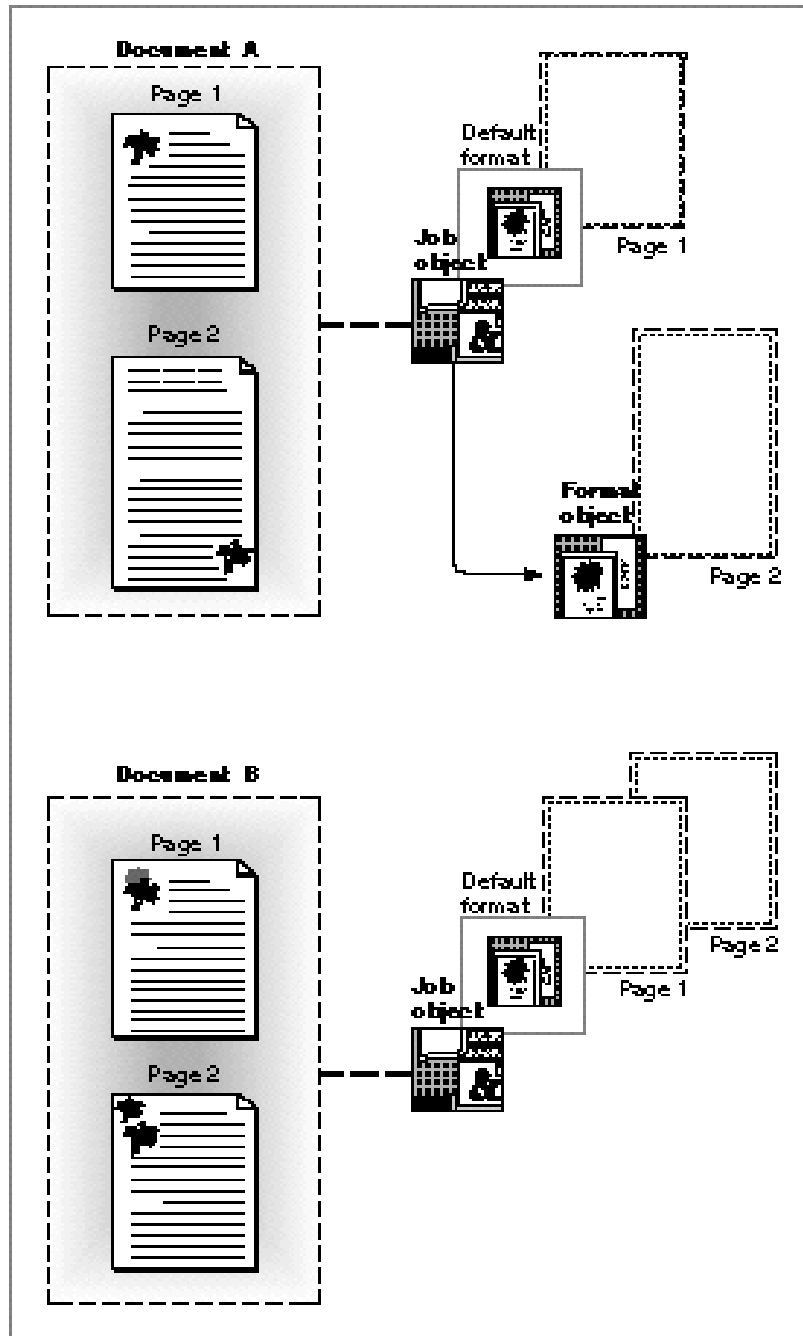
To provide halftone information for shape objects drawn with the same ink, you use a halftone synonym. For detailed information on how to use halftone synonyms, see the chapter “Advanced Printing Features” in this book.

## Copying a Format Object for Use in Other Documents

---

When a user wants to disassociate a format from a particular document and associate it with another document, you use the `GXNewFormat`, `GXCopyFormat`, and `GXDisposeFormat` functions. For example, a user may have a three-page document that contains a format object for a single page in landscape orientation. This user may want to use the landscape page in another document and delete it from the original document.

Figure 3-17 shows two documents. Document A consists of two pages—one page uses the default format, the other uses a unique format object. Document B also consists of two pages—each page uses the default format. A user may decide to use the format of page 2 in Document A for page 1 of Document B.

**Figure 3-17** Moving a format object from one document to another

**Listing 3-14** shows how to move a format object from one document to another. The `srcPage` and `srcDocument` parameters to the `MyMoveFormatToJob` function in the listing represent the page's location in the original document. The `destPage` and `destDocument` parameters refer to the new location and document. Initially, a format object for the destination page does not exist.

---

**Listing 3-14** Moving a format object from one document to another

```
OSErr MyMoveFormatToJob(long srcPage, MyDocumentPtr srcDocument,
                        long destPage, MyDocumentPtr destDocument)
{
    OSErr      err;
    gxFormat    srcPgFormat, destPgFormat;

    /*
       Get the source format object. If it is nil, create a
       destination format object from the source job object.
    */
    srcPgFormat = srcDocument->pageFormat[srcPage-1];
    if (srcPgFormat == nil)
        srcPgFormat = GXNewFormat(srcDocument->documentJob);

    /*
       Create a new destination format object and copy the source
       format object to it. Then dispose of the source format
       object and clear out the source page's reference.
    */
    destPgFormat = GXNewFormat(destDocument->documentJob);
    GXCopyFormat(srcPgFormat, destPgFormat);
    GXDisposeFormat(srcPgFormat);
    srcDocument->pageFormat[srcPage-1] = nil;

    /*
       If there were no errors, store the destination page's format
       object reference.
    */
    err = GXGetJobError(srcDocument->documentJob);
    if (err == noErr)
        err = GXGetJobError(destDocument->documentJob);
    if (err == noErr)
        destDocument->pageFormat[destPage-1] = destPgFormat;
    return err;
}
```

## Obtaining the Mapping From a Format Object

---

To access a format object's mapping, your application uses the `GXGetFormatMapping` function. Listing 3-15 shows how to obtain the mapping for the format object associated with the `whichPage` page.

**Listing 3-15** Obtaining a format object's mapping

---

```
OSErr MyGetFormatMapping(MyDocumentPtr myDocument, long whichPage,
                        gxMapping *theMapping)
{
    gxFormat    pgFormat;

    /*
     * Get the current page's format. A nil reference specifies
     * using the job's format object.
     */
    pgFormat = myDocument->pageFormat[whichPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /* Get the format's mapping. */
    GXGetFormatMapping(pgFormat, theMapping);
    return GXGetJobError(myDocument->documentJob);
}
```

For an introduction to mapping, see “Mapping for Format Objects” beginning on page 3-18.

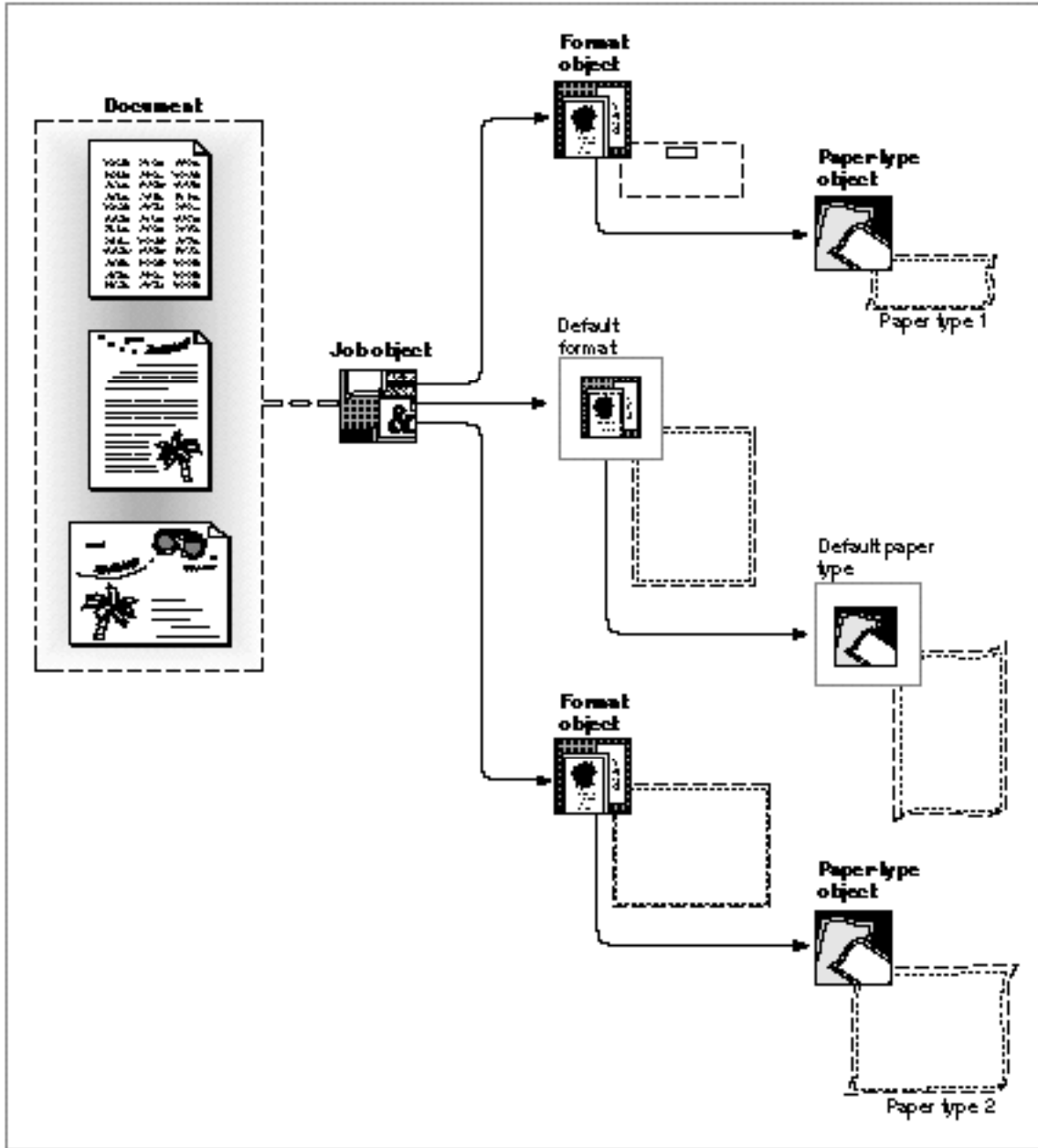
## Obtaining a Paper-Type Object Associated With a Format

---

QuickDraw GX allows a user to specify a paper-type name for each page of a document. Pages with different imageable areas require different format objects. Imageable areas differ both because of physical characteristics (paper size and page size) and because of rendering characteristics (such as scaling and orientation).

Pages require different paper-type objects only when the physical characteristics differ. A change in the paper-type object requires a change in the format object. The job object in Figure 3-18 references three format objects and three paper-type objects. This allows a user to print the address page on an envelope, a letter that contains graphics on an 8.5-by-11 inch sheet of paper in portrait orientation, and a page of graphics on a sheet of paper in landscape orientation.

**Figure 3-18** A three-page document and its corresponding job object, format objects, and paper-type objects



You can use the `GXGetFormatPaperType` function to obtain a format object's associated paper-type object. For detailed information on working with paper-type objects, see the chapter "Advanced Printing Features" in this book.



Listing 3-16 shows how to obtain the paper-type object that a format object references. The `MyGetPaperTypeName` function in the listing returns the name stored in the paper-type object.

---

**Listing 3-16** Obtaining the paper-type object associated with a format object

```
OSErr MyGetPaperTypeName(MyDocumentPtr myDocument, Str255
                        paperTypeName)
{
    gxPaperType    thePaperType;
    long           curPage;
    gxFormat       pgFormat;

    /*
     * Get the current page's format. A nil reference specifies
     * using the job's format object.
     */
    curPage = myDocument->curPage;
    pgFormat = myDocument->pageFormat[curPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /* Get the format's object paper-type object and name. */
    thePaperType = GXGetFormatPaperType(pgFormat);
    GXGetPaperTypeName(thePaperType, paperTypeName);
    return GXGetJobError(myDocument->documentJob);
}
```

## Scanning Through a Job's Format Objects

---

QuickDraw GX allows you to scan through the format objects associated with a particular job. You can use the `GXCountJobFormats` function to obtain the number of format objects in a particular job object. If you want to examine or manipulate each format object for a job, you can use the `GXForEachJobFormatDo` function.

### Note

You cannot use the `GXForEachJobFormatDo` function to modify the default format. u

Listing 3-17 shows the `GXForEachJobFormatDo` function being called to execute the `MyCheckMappingProc` function on each format object.

---

**Listing 3-17** Using the `GXForEachJobFormatDo` function

```
OSErr MyCheckAllFormatMappings(MyDocumentPtr myDocument)
{
    gxMapping    theMapping;

    /* Loop through each format, and check its mapping. */

    GXForEachJobFormatDo(myDocument->documentJob,
                        MyCheckMappingProc, (void *) &theMapping);

    return GXGetJobError(myDocument->documentJob);
}
```

The `GXForEachJobFormatDo` function passes a pointer to the application-supplied function to execute and a pointer to the information that the application-supplied function returns. The prototype for the application-supplied function is as follows:

```
gxLoopStatus MyFormatFunction (gxFormat aFormat, void *refCon);
```

The first parameter, `aFormat`, is a reference to a format object. QuickDraw GX sets this parameter as it calls the function for each format object referenced by a job object. The second parameter, `refCon`, is a pointer to a reference constant through which data can be passed. The return value, `gxLoopStatus`, specifies whether the application-supplied function should be called again, allowing you to terminate the `GXForEachJobFormatDo` function early.

Listing 3-18 shows the application-supplied function, `MyCheckMappingProc`, that obtains scaling and orientation information for each format object associated with a particular job. For example, you can use this function to obtain scaling information when you need to adjust a ruler.

**Listing 3-18** Obtaining scaling information on each format object

---

```

pascal gxLoopStatus MyCheckMappingProc(gxFormat aFormat, void
                                         *theMapping)
{
    /*
       Get the mapping for the current format object, check it out,
       and keep looping until all format objects are accessed.
    */
    GXGetFormatMapping(aFormat, (gxMapping *) theMapping);

    /*
       Your application could adjust rulers here, or do some
       other useful thing based on each format object's mapping.
    */
    ...
    return gxKeepLooping;
}

```

**Note**

For information about the `gxMapping` structure that contains scaling and rotation (orientation) information, see the mathematical functions chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. u

## Associating Format Objects With Document Pages

---

Your application is responsible for managing the correspondence between format objects and individual pages in a document. For example, a user may create a document that consists of three pages. Through the Custom Page Setup dialog box, you can allow a user to specify that pages 1 and 2 use portrait orientation and page 3 uses landscape orientation. In this example, you need to store information that pages 1 and 2 use the default format, while page 3 uses a unique format object.

When a user saves a document containing multiple format objects, you need to save format collection information and then flatten the document's job object. In addition, when a user opens a document containing multiple format objects, you need to unflatten its corresponding job object and retrieve the format collection information. Flattening and unflattening a document's corresponding job object is discussed in the chapter "Core Printing Features" in this book.

There are several methods you can use to store formatting information. A common method, shown in this section, is to save the correspondence between format objects and pages in the format collection. Listing 3-19 shows a function that performs this task.

---

**Listing 3-19** Saving the correspondence between format objects and document pages in a format collection

```
OSErr MySaveFormatRefs(MyDocumentPtr myDocument)
{
    OSErr      err = noErr;
    Handle      theFormatIdxList;
    Collection  fmtCollection;
    gxFormat    defaultFmt;

    /* Create a handle containing all of the format object indices. */
    if (myDocument->numPages > 0)
    {

        /*
         Obtain the format collection. If you have already have a
         document page-to-format object correspondence item stored,
         remove it.
        */
        defaultFmt = GXGetJobFormat(myDocument->documentJob, 1);
        fmtCollection = GXGetFormatCollection(defaultFmt);

        if (fmtCollection != nil)
            RemoveCollectionItem(fmtCollection, kMyFormatInfoType,
                                printingTagID);

        /*
         Create a list of document page-to-format object
         correspondences for the current document. If there are no
         errors, add the item to the format collection for later
         retrieval.
        */
        err = MyCreateFormatIndexList(myDocument,
                                      &theFormatIdxList);
    }
}
```

```

        if (err == noErr)
        {
            HLock(theFormatIdxList);
            err = AddCollectionItem(fmtCollection, kMyFormatInfoType,
                                   printingTagID,
                                   GetHandleSize(theFormatIdxList),
                                   *theFormatIdxList);
            DisposHandle(theFormatIdxList);
        }
    }
    return err;
}

```

The `MyCreateFormatIndexList` function stores the index of each page's format object in a handle. The index of the format object for page 1 goes in the first long word of the `theFormatIdxList` handle. The index of the next page's format object goes in the next long word, and so on. The handle is created and returned to the caller. Listing 3-20 shows the `MyCreateFormatIndexList` function.

---

**Listing 3-20**     Filling the handle

```

OSErr MyCreateFormatIndexList(MyDocumentPtr myDocument, Handle
                               *theFormatIdxList)
{
    OSErr      err;
    long       fmtIdx, pg, *idxList;
    gxFormat   curFormat;

    /*
     * Create a handle large enough to hold all of our entries. This
     * example uses NewHandleClear so that all of our indices are
     * initialized to 0 (an invalid format index). This application
     * stores a nil format reference for each page which uses the
     * default format. This allows us to indicate these "nil
     * references" by an index of 0 in our resource.
     */
    *theFormatIdxList = NewHandleClear(sizeof(long) *
                                       (myDocument->numPages));

    err = MemError();
}

```

## Page Formatting and Dialog Box Customization

```

/*
    If there aren't any errors, go through each format object. If
    the format object is used by any pages of the document, store
    the format object's index in those page entries of
    theFormatIdxList. Skip format object #1, because that's the
    default format.
*/
if (err == noErr)
{
    HLock(*theFormatIdxList);
    idxList = (long *) **theFormatIdxList;

    for (fmtIdx = 2; fmtIdx <=
        GXCountJobFormats(myDocument->documentJob); fmtIdx++)
    {
        curFormat = GXGetJobFormat(myDocument->documentJob,
            fmtIdx);

        for (pg = 1; pg <= myDocument->numPages; pg++)
            if (myDocument->pageFormat[pg - 1] == curFormat)
                idxList[pg - 1] = fmtIdx;
    }
    HUnlock(*theFormatIdxList);
}
return err;
}

```

Listing 3-21 shows how to retrieve format object correspondence from a format collection when a user opens a document containing multiple format objects. This function associates new format object references with a document, based upon the format object indices that are saved with the document. The function is called when a document is opened. The format object references are stored in the passed `MyDocumentRec` structure.

---

**Listing 3-21** Retrieving the correspondence between document pages and format objects from a format collection

```

OSErr MyAdjustFormats(MyDocumentPtr myDocument)
{
    OSErr      err = noErr;
    Handle      theFormatIdxList = nil;
    gxFormat    theFormat, defaultFmt;
    long        pg, numPages, fmtIdx, *idxList, idx, listSize,
               attrs;
    Collection  fmtCollection;

    /*
       Get the format collection, and search for one of the
       document page-to-format object correspondence items.
    */
    defaultFmt = GXGetJobFormat(myDocument->documentJob, 1);
    fmtCollection = GXGetFormatCollection(defaultFmt);

    /*
       Load the item containing the correspondences. First,
       determine if the item exists. Next, create a handle to
       hold the item, and retrieve it. Because there is one
       long-word entry for each page of the document, determine
       the number of pages in the document.
    */
    err = GetCollectionItemInfo(fmtCollection, kMyFormatInfoType,
                               gxPrintingTagID, &idx, &listSize, &attrs);
    if (err == noErr)
        theFormatIdxList = NewHandle(listSize);
    if (theFormatIdxList != nil)
    {
        HLock(theFormatIdxList);
        err = GetCollectionItem(fmtCollection, kMyFormatInfoType,
                               gxPrintingTagID, nil, *theFormatIdxList);
        numPages = listSize / sizeof(long);
    }
}

```

```

/*
    Loop through each saved index. In this example, the first
    index is for page 1, the second is for page 2, and so on.
    Call the GXGetJobFormat function for each saved
    index. Store the format references as they are
    processed. When finished, throw away the handle.
*/
idxList = (long *) *theFormatIdxList;
for (pg = 1; pg <= numPages; pg++)
{
    fmtIdx = idxList[pg - 1];
    if (fmtIdx != nil)
        theFormat = GXGetJobFormat(myDocument->documentJob,
                                    fmtIdx);

    else
        theFormat = nil;
    myDocument->pageFormat[pg - 1] = theFormat;
}
DisposHandle(theFormatIdxList);
}
return err;
}

```

## Customizing QuickDraw GX Dialog Boxes

---

Your application can customize QuickDraw GX dialog boxes. To customize a QuickDraw GX dialog box, you need to take the following general steps:

1. Install a message handler to override the message that causes a QuickDraw GX print dialog box to be displayed. Your override function loads your panel.
2. Create an item list ('DITL') resource that defines the items, such as radio buttons, editable text fields, checkboxes, and pop-up menus, that you want to include in your panel. You may have to create additional resources, such as a control ('CNTL') resource for pop-up menus.
3. Create an icon resource that is displayed in the extended dialog box.
4. Create a panel (gxPrintPanelType) resource that provides a name for your panel and associates the panel with the item list resource and the icon resource.
5. Install a handler to respond to events while the panel is active. This handler can be an extended item list (gxExtendedDITLType) resource or may be an override of the gxHandlePanelEvent message.



These steps need not be done in order; however, all must be completed. The following sections describe how your application adds a panel to a QuickDraw GX dialog box and how to automate the response to user actions using the extended item list (`gxExtendedDITLType`) resource.

## Adding Panels to Dialog Boxes

---

To add a panel to a dialog box, you call the `GXInstallApplicationOverride` function to override the messages that QuickDraw GX sends to display its dialog boxes. The following call to `GXInstallApplicationOverride` sets up the `MyFormatDialogOverride` function to be called when the application receives the `gxFormatDialog` message:

```
GXInstallApplicationOverride(myDocument->documentJob,
                             gxFormatDialog, MyFormatDialogOverride);
```

The `MyFormatDialogOverride` function that is called in response to the message is as follows:

```
OSErr MyFormatDialogOverride(gxFormat aFormat, StringPtr title,
                             gxDialogResult *result)
{
    OSErr    err = noErr;

    err = MySetUpByPagePanel(aFormat, GXGetMessageHandlerResFile());
    if (!err) err = Forward_FormatDialog(aFormat, title, result);
    return err;
}
```

### Note

To remove the application override when a change to the default behavior associated with the message is no longer desired, use the `GXInstallApplicationOverride` function with the function pointer set to `nil` to take the override out of the message chain. u

Because you have specified a function pointer in the `GXInstallApplicationOverride` function to override the message that displays the dialog box, QuickDraw GX calls the `MyFormatDialogOverride` function just before it displays the Custom Page Setup dialog box. The `MyFormatDialogOverride` function calls the `GXSetupDialogPanel` function for each panel that you want to add. The `MyFormatDialogOverride` function must forward the message to the next handler in the message chain.

Listing 3-22 shows the `MySetUpByPagePanel` function, which obtains information from the collection to set up a new panel, calls `GXSetupDialogPanel`, and forwards the message.

---

**Listing 3-22**    Setting up a new panel

```

#define kCreator          'Ex#9'      /* registered
                                     application
                                     creator */
#define kMyKindaCollectionType  kCreator /* collection
                                     tag type */
#define r_MyFormatPanelResID  6000    /* ID of the panel
                                     and panel icon
                                     resources */

typedef struct MyKindaCollectionRec {
    unsigned char  isEnabled;          /* Enabled? */
    char           fillByte;           /* C adds this (if
                                     you don't) for
                                     alignment */
} MyKindaCollectionRec, *MyKindaCollectionPtr,
                        **MyKindaCollectionHdl;

...

OSErr MySetUpByPagePanel(gxFormat aFormat, short ourResFile)
{
    OSErr          err;
    Collection      fmtCollection;
    gxPanelSetupRecord  panelInfo;
    MyKindaCollectionRec  mySettings;

    /*
       Access the format collection and search for the collection
       object item in which the default settings are stored.
    */
    fmtCollection = GXGetFormatCollection(aFormat);

    err = GetCollectionItem(fmtCollection, kMyKindaCollectionType,
                           gxPrintingTagID, nil, &mySettings);

```

## Page Formatting and Dialog Box Customization

```

/*
    If the collection object item does not exist, create one and
    add it to the format collection to support default settings
    for the dialog panel.
*/
if (err == collectionItemNotFoundErr)
{
    mySettings.isEnabled = false;

    err = AddCollectionItem(fmtCollection,
                           kMyKindaCollectionType,
                           gxPrintingTagID,
                           sizeof(MyKindaCollectionRec),
                           &mySettings);
}

/*
    Install the panel. Specify its type, resource ID, and the
    resource file in which it is located.
*/
if (!err)
{
    panelInfo.panelKind      = gxApplicationPanel;
    panelInfo.panelResId     = r_MyFormatPanelResID;
    panelInfo.resourceRefNum = ourResFile;
    panelInfo.refCon         = 0; /* not being used here */
    err = GXSetupDialogPanel(&panelInfo);
}
return err;
}

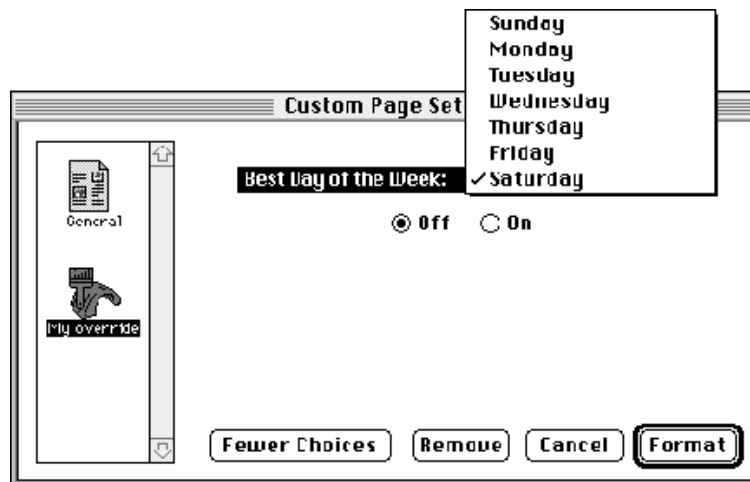
```

Once the user confirms or cancels the dialog box, QuickDraw GX disposes of all panel information. Note that while QuickDraw GX uses a resource file number supplied by the panel owner to look for panel resources, it does not leave the resource chain set to this file. The resource chain's current file is restored once the resources are retrieved.

## Setting Up Dialog Box Resources

Figure 3-19 shows the panel that is loaded in Listing 3-22. Listing 3-23 through Listing 3-27 show the resources required to add this panel.

**Figure 3-19** A panel added to the Custom Page Setup dialog box



Listing 3-23 shows the panel resource, which is added to the dialog box by the `MySetUpByPagePanel` function shown in Listing 3-22.

**Listing 3-23** Sample panel resource

```
#define r_dayPopUpCtl    150    /* ID of the panel's pop-up CNTL */
#define r_dayPopUpMenu  160    /* ID of the panel's pop-up menu */

/* Description of panel added to dialog box. */

resource gxPrintPanelType (r_MyFormatPanelResID, sysheap,
                          purgeable)
{
    "My override", smRoman, r_MyFormatPanelResID, /* Icon ResID */
                          r_MyFormatPanelResID /* Panel ResID */
};
```

Listing 3-24 shows the item list resource, 'DITL', that defines the contents of the panel.

---

**Listing 3-24** Sample item list resource

```
resource 'DITL' (r_MyFormatPanelResID, sysheap, purgeable) {
    {
        {42, 120, 60, 166},
        RadioButton {
            enabled,
            "Off"
        },
        {42, 175, 60, 220},
        RadioButton {
            enabled,
            "On"
        },
        {14, 27, 35, 323},          /* represents the days of the week
                                   pop-up menu */
        Control {
            enabled,
            r_dayPopUpCtl
        }
    }
};
```

**Note**

When you design your 'DITL' resources, note that (0.0, 0.0) for a panel is at the top-left corner of the panel and not at the top-left corner of the dialog box. When you want to locate the position of the cursor within a panel, you use the `GXGetJobPanelDimensions` function to obtain the dimensions of a panel. u

Listing 3-25 shows the control resource, 'CNTL', that defines the pop-up menu control.

---

**Listing 3-25** Sample 'CNTL' resource

```
resource 'CNTL' (r_dayPopUpCtl, sysheap, purgeable)
{
    {72, 4, 93, 300},
    popupTitleLeftJust,      /* menu's title is left justified */
    visible,                  /* show it */
    140,                      /* width of the menu title */
    r_dayPopUpMenu,          /* resource ID of the associated
                             menu */
    popupMenuCDEFproc + popupFixedWidth, /* type of pop-up
                                         menu */
    0,                        /* reference constant */
    "Best Day of the Week:"   /* control's title */
};
```

Listing 3-26 shows the extended item list resource that specifies how to process the items in the panel. For more information about extended item list resources, see “Automating Panel Events” beginning on page 3-25.

---

**Listing 3-26** Sample extended item list resource

```
#define kCreator                'Ex#9'
...
#define kMyKindaCollectionType  kCreator
#define kMyKindaCollectionTagID gxPrintingTagID +1
...
resource gxExtendedDITLType (r_MyFormatPanelResID,
                             sysheap, purgeable)
{
    {
        RadioButtons {kMyKindaCollectionType,
                      kMyKindaCollectionTagID, 0, {1,2}},
        PopUp {kMyKindaCollectionType,
               kMyKindaCollectionTagID, 2, 3}
    };
};
```

This extended item list resource handles two items, a pair of radio buttons, corresponding to the first two items in the 'DITL' resource, and a pop-up menu. All of these items are stored in one collection item, which is identified by the

`kMyKindaCollectionType` collection tag and the `kMyKindaCollectionTagID` item ID. The application creator is used for the collection type to distinguish it from collection items provided by QuickDraw GX. The collection item ID is simply derived from a base; in this case, `gxPrintingTagID`.

The status of the radio buttons occupy the first 2 bytes of the collection item (from offset 0). These bytes specify the status of items 1 and 2. The status of the pop-up menu is at offset 2. It specifies the status of item 3.

Listing 3-27 shows the 'MENU' resource, which specifies the entries in the pop-up menu. Note that the default entry is specified in the collection item.

---

**Listing 3-27** Sample 'MENU' resource

```
resource 'MENU' (r_dayPopUpMenu, sysheap, purgeable) {
    r_dayPopUpMenu,
    textMenuProc,
    allEnabled,
    enabled,
    "",
    {
        "Sunday",      noIcon, noKey, noMark, plain,
        "Monday",      noIcon, noKey, noMark, plain,
        "Tuesday",     noIcon, noKey, noMark, plain,
        "Wednesday",   noIcon, noKey, noMark, plain,
        "Thursday",    noIcon, noKey, noMark, plain,
        "Friday",      noIcon, noKey, noMark, plain,
        "Saturday",    noIcon, noKey, noMark, plain
    }
};
```

## Parsing Page Ranges

---

You can install an override function for the `gxParsePageRange` message, which allows you to check the validity of page numbers that the user selects in the Print dialog box. You must override this message if you allow the user to specify application-specific page ranges, such as “Chapter 5.”

Listing 3-28 shows a function, `MyPrintDialog`, which is called in response to the user choosing the Print menu item from the File menu. The `MyPrintDialog` function installs an override for the `gxParsePageRange` message, sets up a default page range, and calls the `GXPrintDialog` function to display the dialog box with the default page range. After the pages have been printed, or if an error occurred while setting up the default page range, the override function for the `gxParsePageRange` message is removed.

**Listing 3-28** Installing an override function for the `gxParsePageRange` message

---

```

OSError MyPrintDialog(MyDocumentPtr myDocument)
{
    OSError          err;
    gxDialogResult    result;
    gxEditMenuRecord  editMenuRec;

    /* Install an override function to parse page ranges. */
    GXInstallApplicationOverride(myDocument->documentJob,
                                gxParsePageRange,
                                MyParsePageRangeOverride);

    .
    .
    .
    err = MySetupDefaultPageRange(myDocument); /* not shown */
    nrequire(err, CouldNotConfigurePageRange);

    /*
       Display the Print dialog box. If there are no errors and
       the user selects the "OK" button, call a printing routine
       to output the pages.
    */
    result = GXJobPrintDialog(myDocument->documentJob,
                              &editMenuRec);
    err = GXGetJobError(myDocument->documentJob);
    if ((err == noErr) && (result == gxOKSelected))
        err = MyPrintDocument(myDocument); /* not shown */
    .
    .
    .

    /* Remove the parse page range override function. */
    CouldNotConfigurePageRange:
    GXInstallApplicationOverride(myDocument->documentJob,
                                gxParsePageRange, nil);

    return err;
}

```

The `MySetupDefaultPageRange` function that sets up a page range is not shown. For examples of setting up page ranges, see “Specifying Page Ranges in the Job Collection” on page 3-33. The `MyPrintDocument` function that prints pages is not shown. For information about printing pages, see the chapter “Core Printing Features” in this book.



Listing 3-29 shows the override function, `MyParsePageRangeOverride`, which calls another function, `MyPageRangeValidityCheck`, to validate the page range.

**Listing 3-29**    Override function for the `gxParsePageRange` message

---

```
OSErr MyParsePageRangeOverride(StringPtr fromString,
                               StringPtr toString, gxParsePageRangeResult *result)
{
    /*
     * Determine if the "To page" and "From page" strings are
     * valid. If not, the MyPageRangeValidityCheck routine
     * returns gxRangeBadFromValue or gxRangeBadToValue.
     * Otherwise it will return gxRangeParsed.
     */
    if (*result == gxRangeNotParsed)
        *result = MyPageRangeValidityCheck(fromString, toString);

    return noErr;
}
```

The `MyPageRangeValidityCheck` function is not shown. It returns `gxRangeParsed` if the page range is valid, otherwise it returns `gxRangeBadFromValue` if the From value is invalid or `gxRangeBadToValue` if the To value is invalid. For information about parse page range constants, see “The Panel Setup Structure” on page 3-101.

## Page Formatting and Dialog Box Customization Reference

---

This section describes the constants, data types, functions, and resources that are specific to the page formatting and dialog box customization features of QuickDraw GX.

There are several sections that describe constants and data types. The following section, “Constants for Loop Status Information,” describes the constants that can be used when looping over printing-related objects. The section “Constants for Collection Item Categories and Tag IDs” on page 3-76 describes constants for manipulating printing-related collection objects. The section “Constants and Data Types for Job Collection Items” on page 3-78 describes constants and data types for job collections. The section “Constants and Data Types for Format Collection Items” on page 3-89 describes constants and data types for format collections. The section “Constants and Data Types for Paper-Type Collection Items” on page 3-94 describes constants and data types for paper-type collections.

The “Functions” section describes functions for creating and manipulating format objects, manipulating format object properties, displaying the Custom Page Setup dialog

box, obtaining information on a document's format objects, customizing QuickDraw GX dialog boxes, and accessing printing-related collection objects.

The "Application-Defined Functions" section describes message override functions for customizing QuickDraw GX dialog boxes and a function for looping through QuickDraw GX format objects associated with a particular job object.

The "Resources" section describes the panel and extended item list resources used to implement QuickDraw GX dialog boxes.

## Constants for Loop Status Information

---

QuickDraw GX allows you to loop through the printing-related objects associated with another object. For example, QuickDraw GX allows you to loop through the format objects associated with a job object.

To allow you to loop through printing-related objects, QuickDraw GX defines loop status values in the loop status enumeration:

```
enum {
    gxStopLooping = false,
    gxKeepLooping = true
};

typedef Boolean gxLoopStatus;
```

### Constant descriptions

<code>gxStopLooping</code>	If returned, QuickDraw GX stops looping through the specified printing-related objects.
<code>gxKeepLooping</code>	If returned, QuickDraw GX keeps looping through the specified printing-related objects.

## Constants for Collection Item Categories and Tag IDs

---

This section describes the constants provided by QuickDraw GX to manipulate printing-related collections. You can use the collection tag category enumeration to determine collection item data to discard when a printer-driver switch occurs. You can use the collection tag ID enumeration to define collection objects for use with QuickDraw GX printing features.

### Collection Item Categories

---

QuickDraw GX assigns collection object items to several collection item categories. QuickDraw GX tag categories are defined in the collection tag category enumeration, represented by `gxCollectionCategory`:

```
typedef short gxCollectionCategory;

enum {
    gxNoCollectionCategory          = (gxCollectionCategory) 0x0000,
    gxOutputDriverCategory          = (gxCollectionCategory) 0x0001,
    gxFormattingDriverCategory      = (gxCollectionCategory) 0x0002,
    gxDriverVolatileCategory        = (gxCollectionCategory) 0x0004,

    gxVolatileOutputDriverCategory =
        gxOutputDriverCategory + gxDriverVolatileCategory,
    gxVolatileFormattingDriverCategory =
        gxFormattingDriverCategory + gxDriverVolatileCategory
};
```

**Constant descriptions**

gxNoCollectionCategory

The item persists whether or not a printer-driver switch occurs or the collection is flattened.

gxOutputDriverCategory

The item is affected by a change in the output printer driver.

gxFormattingDriverCategory

The item is affected by a change in the formatting printer driver.

gxDriverVolatileCategory

The item is affected by a change in either the output printer driver or formatting printer driver. The item is purged when the collection is flattened if the collectionPersistenceBit is also set.

gxVolatileOutputDriverCategory

The item is purged if the output printer driver changes.

gxVolatileFormattingDriverCategory

The item is purged if the formatting printer driver changes.

## Collection Tag ID

---

QuickDraw GX assigns its collection objects with the same 4-byte collection tag ID. The QuickDraw GX collection tag ID is defined in the collection tag ID enumeration:

```
enum { gxPrintingTagID = -28672 };
```

Collection tag IDs for QuickDraw GX collection objects are discussed in “About Collection Objects,” which begins on page 3-7.

## Constants and Data Types for Job Collection Items

---

The sections that follow identify all of the collection items that QuickDraw GX provides for the job collection object.

### Print-Job Information

---

The collection item ID for print-job information is defined in the following enumeration:

```
enum { gxJobTag = 'job ' };
```

QuickDraw GX stores print-job information in the `gxJobInfo` structure:

```
struct gxJobInfo {
    long    numPages;
    long    priority;
    long    timeToPrint;
    long    jobTimeout;
    long    firstPageToPrint;
    short   jobAlert;
    Str31   appName;
    Str31   documentName;
    Str31   userName;
};
```

#### Field descriptions

<code>numPages</code>	The total number of pages to print. The user specifies the page range to print in the Print dialog box.
<code>priority</code>	The print job's priority. Priorities for print jobs are defined in the print-job priorities enumeration. The user specifies the priority for a print job in the Print Time panel.
<code>timeToPrint</code>	The designated time to print a print job. The user specifies a designated printing time in the Print Time panel.
<code>jobTimeout</code>	The time to cancel the print job, in ticks. QuickDraw GX defines two print-job cancellation times in the print-job cancellation enumeration.
<code>firstPageToPrint</code>	The first page to begin printing.
<code>jobAlert</code>	When to alert the user about printing. QuickDraw GX defines print job alerts in the print-job alert enumeration.
<code>appName</code>	A string containing the name of the application used to create the printable document.
<code>documentName</code>	A string containing the name of the user's document.
<code>userName</code>	A string containing the name of the user associated with the printable document.

QuickDraw GX defines priorities for print jobs in the print-job priorities enumeration:

```
enum {
    gxPrintJobUrgent    = 0x00000001,
    gxPrintJobAtTime    = 0x00000002,
    gxPrintJobASAP      = 0x00000003
};
```

#### Constant descriptions

gxPrintJobUrgent

If set, QuickDraw GX designates a print job as “urgent.”

gxPrintJobAtTime

If set, QuickDraw GX designates the time to print a print job.

gxPrintJobASAP

If set, QuickDraw GX designates a print job as “as soon as possible.”

A holding bit for print-job priorities is defined in the following enumeration:

```
enum { gxPrintJobHoldingBit = 0x00001000 };
```

QuickDraw GX defines holding status for print jobs in the holding status enumeration:

```
enum {
    gxPrintJobHolding          = (gxPrintJobHoldingBit +
                                gxPrintJobASAP),
    gxPrintJobHoldingAtTime    = (gxPrintJobHoldingBit +
                                gxPrintJobAtTime),
    gxPrintJobHoldingUrgent    = (gxPrintJobHoldingBit +
                                gxPrintJobUrgent)
};
```

#### Constant descriptions

gxPrintJobHolding

If set, QuickDraw GX assigns a print job designated as “as soon as possible” to a holding status.

gxPrintJobHoldingAtTime

If set, QuickDraw GX assigns a print job designated to print at a specific time to a holding status.

gxPrintJobHoldingUrgent

If set, QuickDraw GX assigns a print job designated as “urgent” to a holding status.

QuickDraw GX defines print job alerts in the print-job alert enumeration:

```
enum {
    gxNoPrintTimeAlert= 0,
    gxAlertBefore      = 1,
    gxAlertAfter       = 2,
    gxAlertBothTimes   = 3
};
```

#### Constant descriptions

gxNoPrintTimeAlert

If set, QuickDraw GX doesn't alert the user about printing.

gxAlertBefore If set, QuickDraw GX alerts the user that printing is about to begin.

gxAlertAfter If set, QuickDraw GX alerts the user that printing has finished.

gxAlertBothTimes

If set, QuickDraw GX alerts the user when printing begins and finishes.

QuickDraw GX defines two print-job cancelation times in the print-job cancelation enumeration, which you could use if the user failed to respond to a condition, such as out of paper:

```
enum {
    gxThirtySeconds = 1800,
    gxTwoMinutes    = 7200
};
```

#### Constant descriptions

gxThirtySeconds

If set, QuickDraw GX cancels a print job in 30 seconds, or 1800 ticks.

gxTwoMinutes

If set, QuickDraw GX cancels a print job in 2 minutes, or 7200 ticks.

## Collation Information

---

The collection item ID for collation information is defined in the following enumeration:

```
enum { gxCollationTag = 'sort' };
```

QuickDraw GX stores collation information in the collation information structure:

```
struct gxCollationInfo {
    Boolean collation;
};
```

**Field descriptions**

`collation` A Boolean value indicating whether the user wants to collate document pages when printed. When the user chooses the Collate Copies checkbox in the Print dialog box, the `collation` field contains `true`; otherwise, the field contains `false`.

## Copies Information

---

The collection item ID for copies information is defined in the following enumeration:

```
enum { gxCopiesTag = 'copy' };
```

QuickDraw GX stores copies information in the copies information structure:

```
struct gxCopiesInfo {
    long copies;
};
```

**Field descriptions**

`copies` The number of copies of a document to print. A user specifies the number of copies to print in the Print dialog box.

The Print dialog box is discussed in the chapter “Core Printing Features” in this book.

## Page-Range Information

---

The collection item ID for page-range information is defined in the following enumeration:

```
enum { gxPageRangeTag = 'rang' };
```

QuickDraw GX stores page-range information in the `gxPageRangeInfo` structure:

```
struct gxPageRangeInfo {
    gxSimplePageRangeInfo simpleRange;
    Str31 fromString;
    Str31 toString;
    long minFromPage;
    long maxToPage;
    char replaceString[1];
};
```

**Field descriptions**

`simpleRange` A string containing the page-range information structure.

`fromString` A string containing the beginning of a user-specified custom page range.

`toString` A string containing the end of a user-specified custom page range.

## Page Formatting and Dialog Box Customization

<code>minFromPage</code>	The minimum default page range.
<code>maxToPage</code>	The maximum default page range.
<code>replaceString</code>	A string containing the user-specified page range from the Print dialog box. Initially, the string is one character long.

QuickDraw GX stores simple page-range information in the `gxSimplePageRangeInfo` structure:

```
struct gxSimplePageRangeInfo {
    char    optionChosen;
    Boolean printAll;
    long    fromPage;
    long    toPage;
};
```

**Field descriptions**

<code>optionChosen</code>	A character that contains the specific page-range option (either the default page range, replacement page range, or customized page range).
<code>printAll</code>	A Boolean value indicating whether the user wants to print all of the pages in a single document. When the user chooses the All radio button in the Print dialog box, the <code>printAll</code> field contains <code>true</code> ; otherwise, the field contains <code>false</code> .
<code>fromPage</code>	The first page in the page range to print. The user specifies a page range to print in the Print dialog box.
<code>toPage</code>	The last page in the page range to print. The user specifies a page range to print in the Print dialog box.

QuickDraw GX defines page-range options in the following enumeration:

```
enum {
    gxDefaultPageRange    = (char) 0,
    gxReplacePageRange    = (char) 1,
    gxCustomizePageRange  = (char) 2
};
```

**Constant descriptions**

<code>gxDefaultPageRange</code>	If set, QuickDraw GX uses a standard numeric page range; for example, the From field of the Print dialog box contains 1 and the To field contains 4.
<code>gxReplacePageRange</code>	If set, QuickDraw GX uses a single editable text field that specifies a page range; for example, a field with “Chapter 5” as the contents.
<code>gxCustomizePageRange</code>	If set, QuickDraw GX allows alphanumeric values for the From and To fields in the Print dialog box. You are responsible for validation of these values.



## Quality Information

---

The collection item ID for quality information is defined in the following enumeration:

```
enum { gxQualityTag = 'qual' };
```

QuickDraw GX stores quality information in the `gxQualityInfo` structure:

```
struct gxQualityInfo {
    Boolean    disableQuality;
    short      defaultQuality;
    short      currentQuality;
    short      qualityCount;
    char       qualityNames[1];
};
```

### Field descriptions

<code>disableQuality</code>	A Boolean value indicating whether to disable standard quality controls.
<code>defaultQuality</code>	The index of the string that represents the default quality.
<code>currentQuality</code>	The index of the string that represents the current quality.
<code>qualityCount</code>	The number of quality menu items displayed in the Quality pop-up menu in the Print dialog box.
<code>qualityNames</code>	A list of packed strings (1-byte string length preceding the actual string) that contain the menu item names (such as “Best”) displayed in the Quality pop-up menu in the Print dialog box.

## File-Destination Information

---

The collection item ID for file-destination information is defined in the following enumeration:

```
enum { gxFileDestinationTag = 'dest' };
```

QuickDraw GX stores file-destination information in the file-destination information structure:

```
struct gxFileDestinationInfo {
    Boolean toFile;
};
```

**Field descriptions**

**toFile** A Boolean value indicating whether the user wants to print a document to a file. When the user chooses File in the Destination pop-up menu in the Print dialog box, the `toFile` field contains `true`. When the user chooses Printer, the `toFile` field contains `false`.

## File-Location Information

---

The collection item ID for file-location information is defined in the following enumeration:

```
enum { gxFileLocationTag = 'floc' }
```

QuickDraw GX stores file-location information in the `gxFileLocationInfo` structure:

```
struct gxFileLocationInfo {
    FSSpec fileSpec;
};
```

**Field descriptions**

**fileSpec** A file system specification containing the location of the file in which to print the user's document.

## File-Format Information

---

The collection item ID for file-format information is defined in the following enumeration:

```
enum { gxFileFormatTag = 'ffmt' };
```

QuickDraw GX stores file-format information in the `gxFileFormatInfo` structure:

```
struct gxFileFormatInfo {
    Str31 fileFormatName;
};
```

**Field descriptions**

**fileFormatName** A string containing the name of the format in which to print the user's document.

## File-Fonts Information

---

The collection item ID for file-fonts information is defined in the following enumeration:

```
enum { gxFileFontsTag = 'incf' };
```

QuickDraw GX stores file-fonts information in the `gxFileFontsInfo` structure:

```
struct gxFileFontsInfo {
    char    includeFonts;
};
```

### Field descriptions

`includeFonts`     A character that specifies the level of fonts to include when a user prints to a file.

The level of fonts to include are defined by the following enumeration:

```
enum {
    gxIncludeNoFonts           = (char) 1,
    gxIncludeAllFonts          = (char) 2,
    gxIncludeNonStandardFonts = (char) 3
};
```

### Constant descriptions

`gxIncludeNoFonts`  
Do not include any fonts.

`gxIncludeAllFonts`  
Include all fonts.

`gxIncludeNonStandardFonts`  
Do not include standard fonts.

## Paper-Feed Information

---

The collection item ID for paper-feed information is defined in the following enumeration:

```
enum { gxPaperFeedTag = 'feed' };
```

QuickDraw GX stores paper-feed information in the `gxPaperFeedInfo` structure:

```
struct gxPaperFeedInfo {
    Boolean    autoFeed;
};
```

**Field descriptions**

**autoFeed** A Boolean value indicating whether the user wants to use automatic or manual paper feed. When the user chooses the Automatic radio button in the Print dialog box, the `autoFeed` field contains `true`. When the user chooses the Manual radio button in the Print dialog box, the `autoFeed` field contains `false`.

## Manual-Feed Information

---

The collection item ID for manual-feed information is defined in the following enumeration:

```
enum { gxManualFeedTag = 'manf' };
```

QuickDraw GX stores manual-feed information in the `gxManualFeedInfo` structure:

```
struct gxManualFeedInfo {
    long          numPaperTypeNames;
    Str31         paperTypeNames[1];
};
```

**Field descriptions**

**numPaperTypeName**

The number of paper-type objects to manually feed.

**paperTypeNames**

A string containing the names of paper-type objects to manually feed.

## Standard Mapping Information

---

The collection item ID for standard mapping information is defined in the following enumeration:

```
enum { gxNormalMappingTag = 'nmap' };
```

QuickDraw GX stores standard mapping information in the `gxNormalMappingInfo` information structure:

```
struct gxNormalMappingInfo {
    Boolean       normalPaperMapping;
};
```

**Field descriptions****normalPaperMapping**

A Boolean value indicating whether the user wants standard or special paper mapping to print a document. When the user chooses to print by specifying input-tray paper matching in the Paper Match panel, the `normalPaperMapping` field is `true`. When the user chooses to ignore paper matching and redirect the document, the `normalPaperMapping` field is `false`.

## Special Mapping Information

---

The collection item ID for special mapping information is defined in the following enumeration:

```
enum { gxSpecialMappingTag = 'smap' };
```

QuickDraw GX stores special mapping information in the `gxSpecialMappingInfo` structure:

```
struct gxSpecialMappingInfo {
    char          specialMapping;
};
```

**Field descriptions****specialMapping**

A character which specifies how to handle paper matching if the user chooses to ignore paper matching and redirect the document.

The following enumeration specifies the possible paper-mapping options:

```
enum {
    gxRedirectPages    = (char) 1,
    gxScalePages       = (char) 2,
    gxTilePages        = (char) 3
};
```

**Constant descriptions****gxRedirectPages**

If set, QuickDraw GX crops the pages of a redirected document.

**gxScalePages**

If set, QuickDraw GX scales the pages of a document to fit the physical page size.

**gxTilePages**

If set, QuickDraw GX tiles the pages of a document.

## Tray-Mapping Information

---

The collection item ID for tray-mapping information is defined in the following enumeration:

```
enum { gxTrayMappingTag = 'tmap' };
```

The tray-mapping information is defined in a `gxTrayMappingInfo` structure:

```
struct gxTrayMappingInfo {
    gxTrayIndex mapPaperToTray;
};
```

The tray index type is used to designate a specific paper tray on a printer:

```
typedef long gxTrayIndex;
```

## Print-Panel Information

---

The collection item ID for print-panel information is defined in the following enumeration:

```
enum { gxPrintPanelTag = 'ppan' };
```

QuickDraw GX stores print-panel information in the `gxPrintPanelInfo` structure:

```
struct gxPrintPanelInfo {
    Str31    startPanelName;
};
```

### Field descriptions

`startPanelName`

A string containing the name of the first panel to display in the Print dialog box.

## Format-Panel Information

---

The collection item ID for format-panel information is defined in the following enumeration:

```
enum { gxFormatPanelTag = 'fpan' };
```

QuickDraw GX stores format-panel information in the `gxFormatPanelInfo` structure:

```
struct gxFormatPanelInfo {
    Str31    startPanelName;
};
```

**Field descriptions**`startPanelName`

A string containing the name of the first panel to display in the Page Setup dialog box.

## Paper-Mapping Information

---

The collection item ID for paper-mapping information is defined in the following enumeration:

```
enum { gxPaperMappingTag = 'pmap' };
```

This collection item contains the flattened paper type that was selected for redirection.

## Translated-Document Information

---

The collection item ID for translated-document information is defined in the following enumeration:

```
enum { gxTranslatedDocumentTag = 'trns' };
```

QuickDraw GX stores translated-document information in the `gxTranslatedDocumentInfo` structure:

```
struct gxTranslatedDocumentInfo {
    long translatorInfo;
};
```

**Field descriptions**

`translatorInfo` A value that specifies translation information for the document.

## Constants and Data Types for Format Collection Items

---

The sections that follow identify all of the collection items that QuickDraw GX provides for the format collection object.

## Orientation Information

---

The collection item ID for orientation information is defined in the following enumeration:

```
enum { gxOrientationTag = 'layo' };
```

**QuickDraw GX stores orientation information in the `gxOrientationInfo` structure:**

```
struct gxOrientationInfo {
    char    orientation;
} ;
```

#### Field descriptions

**orientation**      A character that contains the orientation information. For example, a user may choose to print a document in portrait, landscape, or rotated landscape orientation.

**QuickDraw GX defines orientation options in the following enumeration:**

```
enum {
    gxPortraitLayout          = (char) 0,
    gxLandscapeLayout         = (char) 1,
    gxRotatedPortraitLayout   = (char) 2,
    gxRotatedLandscapeLayout  = (char) 3
};
```

#### Constant descriptions

**gxPortraitLayout**      If set, QuickDraw GX uses portrait orientation for the user-specified page.

**gxLandscapeLayout**      If set, QuickDraw GX uses landscape orientation for the user-specified page.

**gxRotatedPortraitLayout**      If set, QuickDraw GX uses rotated portrait orientation for the page.

**gxRotatedLandscapeLayout**      If set, QuickDraw GX uses rotated landscape orientation for the user-specified page.



## Scaling Information

---

The collection item ID for scaling information is defined in the scaling information enumeration:

```
enum { gxScalingTag = 'scal' };
```

QuickDraw GX stores scaling information in the `gxScalingInfo` structure:

```
struct gxScalingInfo{
    Fixed    horizontalScaleFactor;
    Fixed    verticalScaleFactor;
    short    minScaling;
    short    maxScaling;
};
```

### Field descriptions

<code>horizontalScaleFactor</code>	The current horizontal scaling factor.
<code>verticalScaleFactor</code>	The current vertical scaling factor.
<code>minScaling</code>	The minimum current scaling factor.
<code>maxScaling</code>	The maximum current scaling factor.

## Direct-Mode Information

---

The collection item ID for direct-mode information is defined in the following enumeration:

```
enum { gxDirectModeTag = 'dirm' };
```

QuickDraw GX stores direct-mode information in the `gxDirectModeInfo` structure:

```
struct gxDirectModeInfo {
    Boolean    directModeOn;
};
```

### Field descriptions

<code>directModeOn</code>	A Boolean value indicating whether the user wants to print using direct mode. When the user chooses the Direct checkbox in the Page Setup dialog box, the <code>directModeOn</code> field contains <code>true</code> ; otherwise, this field contains <code>false</code> .
---------------------------	--

## Format-Halftone Information

---

The collection item ID for halftone information is defined in the halftone information enumeration:

```
enum { gxFormatHalftoneTag = 'half' };
```

QuickDraw GX stores halftone information in the `gxFormatHalftoneInfo` structure:

```
struct gxFormatHalftoneInfo {
    long    numHalftones;
    gxHalftone  halftones[1];
};
```

### Field descriptions

<code>numHalftones</code>	The number of halftones in the structure.
<code>halftones</code>	The halftones to use when rendering a page with this format.

## Page-Inversion Information

---

The collection item ID for page-inversion information is defined in the following enumeration:

```
enum { gxInvertPageTag = 'invp' };
```

QuickDraw GX stores page-inversion information in the `gxInvertPageInfo` structure:

```
struct gxInvertPageInfo {
    Boolean  invert;
};
```

### Field descriptions

<code>invert</code>	The user-specified page-inversion information, which indicates whether a user chooses to invert a page before printing. If <code>true</code> , the page is inverted; otherwise, it is not inverted.
---------------------	---

## Horizontal Page-Flip Information

---

The collection item ID for horizontal page-flip information is defined in the following enumeration:

```
enum { gxFlipPageHorizontalTag = 'flph' };
```

QuickDraw GX stores horizontal page-flip information in the `gxFlipPageHorizontalInfo` structure:

```
struct gxFlipPageHorizontalInfo {
    Boolean flipHorizontal;
};
```

### Field descriptions

`flipHorizontal`

The user-specified horizontal page-flip information. If `true`, a user chooses to horizontally flip the x coordinate on a page before printing.

## Vertical Page-Flip Information

---

The collection item ID for vertical page-flip information is defined in the following enumeration:

```
enum { gxFlipPageVerticalTag = 'flpv' };
```

QuickDraw GX stores vertical page-flip information in the `gxFlipPageVerticalInfo` structure:

```
struct gxFlipPageVerticalInfo {
    Boolean flipVertical;
};
```

### Field descriptions

`flipVertical`

The user-specified vertical page-flip information. If `true`, a user chooses to vertically flip the y coordinate on a page before printing.

## Precise-Bitmap Information

---

The collection item ID for precise-bitmap information is defined in the following enumeration:

```
enum { gxPreciseBitmapsTag = 'pbmp' };
```

QuickDraw GX stores page bitmap information in the `gxPreciseBitmapInfo` structure:

```
struct gxPreciseBitmapInfo {
    Boolean preciseBitmaps;
};
```

#### Field descriptions

`preciseBitmaps`

The user-specified precise-bitmap information. If `true`, a user chooses to scale a page by 96%.

## Paper-Type Lock Information

---

The collection item ID for lock information is defined in the following enumeration:

```
enum { gxPaperTypeLockTag = 'ptlk' };
```

QuickDraw GX stores paper-type object lock information in the `gxPaperTypeLockInfo` structure:

```
struct gxPaperTypeLockInfo {
    Boolean paperTypeLocked;
};
```

#### Field descriptions

`paperTypeLocked`

A Boolean value indicating whether a paper-type object is locked.

## Constants and Data Types for Paper-Type Collection Items

---

The sections that follow identify all of the collection items QuickDraw GX provides for the paper-type collection object.

## Base Information

---

The collection item ID for base information is defined in the following enumeration:

```
enum { gxBaseTag = 'base' };
```

QuickDraw GX stores base information in the `gxBaseInfo` structure:

```
struct gxBaseInfo {
    long baseType;
};
```

**Field descriptions**

**baseType**      The user-specified base information, which indicates whether the source of the paper is unknown, US Letter, US Legal, A4, B5, or tabloid.

QuickDraw GX defines paper-type object base types in the following enumeration:

```
enum {
    gxUnknownBase    = 0,
    gxUsLetterBase   = 1,
    gxUsLegalBase    = 2,
    gxA4LetterBase   = 3,
    gxB5LetterBase   = 4,
    gxTabloidBase    = 5
};
```

**Constant descriptions**

**gxUnknownBase**    If set, the base type is unknown.

**gxUsLetterBase**    If set, QuickDraw GX uses a US Letter base type.

**gxUsLegalBase**    If set, QuickDraw GX uses a US Legal base type.

**gxA4LetterBase**    If set, QuickDraw GX uses an A4 base type.

**gxB5LetterBase**    If set, QuickDraw GX uses a B5 base type.

**gxTabloidBase**    If set, QuickDraw GX uses a tabloid base type.

## Creator Information

---

The collection item ID for creator information is defined in the following enumeration:

```
enum { gxCreatorTag = 'crea' };
```

QuickDraw GX stores paper-type object creator information in the `gxCreatorInfo` structure:

```
struct gxCreatorInfo {
    OSType    creator;
} ;
```

**Field descriptions**

**creator**      An operating-system type that contains the creator type of a paper-type object. You specify a system paper-type object creator as 'sypt', and you specify a user paper-type object creator as 'uspt'.

Applications do not need to set this collection item. Printer drivers that create paper-type objects should use the creator type that identifies the printer driver. For example, a printer driver for the LaserWriter SC should specify a paper-type object creator as 'lwsc'.

QuickDraw GX defines paper-type object creator types in the following enumeration:

```
enum {
    gxSysPaperType      = 'sypt',
    gxUserPaperType     = 'uspt'
};
```

#### Constant descriptions

**gxSysPaperType** If set, QuickDraw GX uses a system-defined paper-type object.

**gxUserPaperType**

If set, QuickDraw GX uses a user-defined paper-type object.

## Units Information

---

The collection item ID for units information is defined in the following enumeration:

```
enum { gxUnitsTag = 'unit' };
```

QuickDraw GX stores units information in the `gxUnitsInfo` structure:

```
struct gxUnitsInfo {
    char    units;
} ;
```

#### Field descriptions

**units** A character that contains the units for a paper-type object. Units can be specified in picas, millimeters, and inches.

QuickDraw GX defines paper-type object units in the following enumeration:

```
enum {
    gxPicas  = (char) 0,
    gxMms    = (char) 1,
    gxInches = (char) 2
};
```

#### Constant descriptions

**gxPicas** If set, QuickDraw GX uses picas to define paper-type object units.

**gxMms** If set, QuickDraw GX uses millimeters to define paper-type object units.

**gxInches** If set, QuickDraw GX uses inches to define paper-type object units.

## Flags Information

---

The collection item ID for flags information is defined in the following enumeration:

```
enum { gxFlagsTag = 'flag' };
```

QuickDraw GX stores flags information in the following structure:

```
struct gxFlagsInfo{
    long  flags;
};
```

### Field descriptions

**flags**                      The flags information for a paper-type object. A flag is a bit position that indicates the system software version used to create a paper-type object.

QuickDraw GX defines paper-type object flags in the following enumeration:

```
enum {
    gxOldPaperTypeFlag      = 0x00800000,
    gxNewPaperTypeFlag      = 0x00400000,
    gxOldAndNewPaperTypeFlag= 0x00C00000,
    gxDefaultPaperTypeFlag  = 0x00100000,
};
```

### Constant descriptions

**gxOldPaperTypeFlag**

A paper type used only with applications that do not support QuickDraw GX printing.

**gxNewPaperTypeFlag**

A paper type used only with applications that do support QuickDraw GX printing.

**gxOldAndNewPaperTypeFlag**

A paper type used with applications that support QuickDraw GX printing and with those that do not.

**gxDefaultPaperTypeFlag**

The default paper type.

## Comment Information

---

The collection item ID for comment information is defined in the following enumeration:

```
enum { gxCommentTag = 'cmnt' };
```

QuickDraw GX stores comment information in the `gxCommentInfo` structure:

```
struct gxCommentInfo {
    Str255    comment;
};
```

#### Field descriptions

<code>comment</code>	A string containing an application-specified comment to associate with a paper-type object.
----------------------	---

## Panel-Related Constants and Data Types

---

The following sections describe the constants and data types related to panels.

### The Panel Information Structure

---

The panel information structure, of data type `gxPanelInfoRecord`, provides information to the panel about the current dialog box and panel event. This structure is used with the `GXHandlePanelEvent` and `GXFilterPanelEvent` override functions, whose descriptions begin on page 3-123.

```
struct gxPanelInfoRecord {
    gxPanelEvent    panelEvt;
    short           panelResId;
    DialogPtr       pDlg;
    EventRecord     *theEvent;
    short           itemHit;
    short           itemCount;
    short           evtAction;
    short           errorStringId;
    gxFormat        theFormat;
    void            *refCon;
};
```

#### Field descriptions

<code>panelEvt</code>	The event to filter or handle.
<code>panelResId</code>	The resource ID of the current panel ( <code>gxPrintPanelType</code> ) resource.
<code>pDlg</code>	A pointer to the dialog box structure.
<code>theEvent</code>	A pointer to the event that occurred.
<code>itemHit</code>	The actual item number where the event occurred, using the item-numbering scheme of the Dialog Manager.
<code>itemCount</code>	The item count before your panel's items in the dialog box.



evtAction	The action that results once this event is processed. This value is one of the constants defined in the panel event actions enumeration, which is described on page 3-101. This field is only meaningful for filtering, and is used for parsing.
errorStringId	The ID of the 'STR ' resource to put in the error alert. A value of 0 in this field indicates that there is no error string to display.
theFormat	The current format. This is only meaningful in a Custom Page Setup dialog box.
refCon	A reference constant for use by the generator of the panel.

## Panel Events

---

The panel event enumeration defines the possible event types that can occur in a panel. This data type is used with the panel information structure, which is described in the previous section.

```
enum {
    gxPanelNoEvt          = (gxPanelEvent) 0,
    gxPanelOpenEvt        = (gxPanelEvent) 1,
    gxPanelCloseEvt        = (gxPanelEvent) 2,
    gxPanelHitEvt          = (gxPanelEvent) 3,
    gxPanelActivateEvt     = (gxPanelEvent) 4,
    gxPanelDeactivateEvt   = (gxPanelEvent) 5,
    gxPanelIconFocusEvt    = (gxPanelEvent) 6,
    gxPanelPanelFocusEvt   = (gxPanelEvent) 7,
    gxPanelFilterEvt       = (gxPanelEvent) 8,
    gxPanelCancelEvt       = (gxPanelEvent) 9,
    gxPanelConfirmEvt      = (gxPanelEvent) 10,
    gxPanelDialogEvt       = (gxPanelEvent) 11,
    gxPanelOtherEvt        = (gxPanelEvent) 12,
    gxUserWillConfirmEvt   = (gxPanelEvent) 13
};

typedef long gxPanelEvent;
```

### Constant descriptions

gxPanelNoEvt	No event has occurred.
gxPanelOpenEvt	The panel is about to open. It needs to be initialized and drawn.
gxPanelCloseEvt	The panel is about to close.
gxPanelHitEvt	The user has selected an item in the panel.
gxPanelActivateEvt	The dialog box in which the panel resides has just been activated.

<code>gxPanelDeactivateEvt</code>	The dialog box in which the panel resides is about to be deactivated.
<code>gxPanelIconFocusEvt</code>	The focus has changed from the panel to the icon list.
<code>gxPanelPanelFocusEvt</code>	The focus has changed from the icon list to the panel.
<code>gxPanelFilterEvt</code>	The panel event needs to be filtered.
<code>gxPanelCancelEvt</code>	The user has selected the Cancel button in the dialog box.
<code>gxPanelConfirmEvt</code>	The user has selected the OK button in the dialog box.
<code>gxPanelDialogEvt</code>	An event has occurred in the panel that is going to be handled by a dialog box handler such as the application, a printing extension, a printer driver, or the Macintosh system software.
<code>gxPanelOtherEvt</code>	A different kind of event, such as an operating-system event, has occurred in the panel.
<code>gxPanelUserWillConfirmEvt</code>	The user has selected the confirm button, which means that it is time to parse panel interdependencies.

## Panel Responses

---

A handler of a panel in a dialog box (including applications, printing extensions, printer drivers, and Macintosh system software) can return any value of type `OSErr` as the result of handling the panel. In addition, a panel handler can return an event of type `gxPanelResult`, as shown here. This data type is used with the `GXHandlePanelEvent` override function, which is described on page 3-123.

```
enum {
    gxPanelNoResult          = 0,
    gxPanelCancelConfirmation = 1
};

typedef long gxPanelResult;
```

### Constant descriptions

<code>gxPanelNoResult</code>	The result field does not currently have any meaning.
<code>gxPanelCancelConfirmation</code>	This result is only valid if the panel event (as described in the previous section) was of type <code>gxPanelUserWillConfirmEvt</code> . After the user confirms the panel, if the panel handler discovers that the user entered an inappropriate value, the panel handler alerts the user to the problem and generates this response, which tells

QuickDraw GX to not confirm the dialog box. This allows the user the opportunity to fix the problem.

## Panel Event Actions

---

The panel event actions enumeration defines the constants used in the `evtAction` field of the panel information structure, which is described on page 3-98. Each value defines what action takes place after an event is processed.

```
enum {
    gxOtherAction          = 0,
    gxClosePanelAction     = 1,
    gxCancelDialogAction   = 2,
    gxConfirmDialogAction  = 3
};
```

### Constant descriptions

`gxOtherAction`     The current item does not change after processing this event.

`gxClosePanelAction`

                    The panel is closed after this event is processed.

`gxCancelDialogAction`

                    The dialog box is canceled after this event is processed.

`gxConfirmDialogAction`

                    The dialog box is confirmed after this event is processed.

## The Panel Setup Structure

---

The panel setup structure, of data type `gxPanelSetupRecord`, is passed to the `GXSetupDialogPanel` function when the user displays a dialog box.

```
struct gxPanelSetupRecord {
    gxPrintingPanelKind   panelKind;
    short                 panelResId;
    short                 resourceRefNum;
    void                  *refCon;
};
```

### Field descriptions

`panelKind`     The kind of program that is using this panel. This value is one of the constants defined in the printing panel kinds enumeration, which is described in the next section.

`panelResId`     The resource ID of the panel ('ppnl') resource for the dialog box panel.

`resourceRefNum`

                    The resource file reference number for the panel.

`refCon`

                    A reference constant for use by the creator of the panel.

## Printing Panel Kinds

---

The printing panel kinds enumeration provides constants for use in the `panelKind` field of the panel setup structure, which is described in the previous section.

```
enum {
    gxApplicationPanel= (gxPrintingPanelKind) 0,
    gxExtensionPanel   = (gxPrintingPanelKind) 1,
    gxDriverPanel      = (gxPrintingPanelKind) 2
};

typedef long gxPrintingPanelKind;
```

### Constant descriptions

<code>gxApplicationPanel</code>	A panel created for an application.
<code>gxExtensionPanel</code>	A panel created for a printing extension.
<code>gxDriverPanel</code>	A panel created for a printer driver.

## Parse Range Results

---

The parse range results enumeration provides the constants that are used to parse dialog box item responses.

```
enum {
    gxRangeNotParsed      = (gxParsePageRangeResult) 0,
    gxRangeParsed         = (gxParsePageRangeResult) 1,
    gxRangeBadFromValue   = (gxParsePageRangeResult) 2,
    gxRangeBadToValue     = (gxParsePageRangeResult) 3
};

typedef long gxParsePageRangeResult;
```

### Constant descriptions

<code>gxRangeNotParsed</code>	QuickDraw GX has not yet parsed a page range in the string.
<code>gxRangeParsed</code>	QuickDraw GX has successfully parsed a page range in the string.
<code>gxRangeBadFromValue</code>	QuickDraw GX has encountered an invalid value in the “from page” string during the parse.
<code>gxRangeBadToValue</code>	QuickDraw GX has encountered an invalid value in the “to page” string during the parse.

## Functions

---

This section describes the functions for creating and manipulating format objects, manipulating format object properties, displaying the Custom Page Setup dialog box, obtaining information on a document's format objects, customizing QuickDraw GX dialog boxes, and accessing printing-related collection objects.

Included with each function description is a list of specific result codes returned by QuickDraw GX. In addition to these result codes, you may also receive file-system, memory, and resource errors. For a complete listing of specific file-system, memory, and resource errors, see *Inside Macintosh: C Summary* or *Inside Macintosh: Pascal Summary*.

You should note that not all possible result codes for a particular function are included in function descriptions within this section. For example, the Message Manager, described in *Inside Macintosh: QuickDraw GX Environment and Utilities*, allows QuickDraw GX functions to send specific messages to your application. These messages can also generate errors.

### IMPORTANT

All printing functions in QuickDraw GX, with the exception of the `GXGetJobError` function, may move Macintosh memory. The `GXGetJobError` function, however, relies on data that may also move. Therefore, your application should never call a QuickDraw GX printing-related function at interrupt time. *s*

## Creating and Manipulating Format Objects

---

When a user creates a new document, clicks on a page, and chooses the Format button in the Custom Page Setup dialog box, you use the `GXNewFormat` function to create a new format object.

When a user wants to modify a format for a single page that is also shared by other pages in the same document, the user wants to return to the default format, or the user decides not to save a format, you use the `GXDisposeFormat` function to dispose of the format object, which decrements its owner count.

When a user wants to disassociate a format from a particular document and associate it with another document, you use the `GXCopyFormat` function to copy a format object. When a user wants to share a format, created using the Custom Page Setup dialog box, with an additional page in the same document, you use the `GXCloneFormat` function to clone a format object. This function increments the owner count.

You can use the `GXCountJobFormats` function to obtain the number of format objects in a particular document, and you can use the `GXForEachJobFormatDo` function to make changes to each format object associated with a printable document. You can use the `GXCountFormatOwners` function to determine the number of references to a format object.

## GXNewFormat

---

You can use the `GXNewFormat` function to create a format object.

```
gxFormat GXNewFormat (gxJob aJob);
```

`aJob`                    A reference to the job object to be associated with the new format object.

*function result*    A reference to a format object.

### DESCRIPTION

The `GXNewFormat` function creates a new format object and copies the default format for the specified job object. The `GXNewFormat` function sets the owner count to 1. You need to call this function each time a user creates a new format for a page in a document.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment for QuickDraw GX printing features failed to load due to low memory or disk errors.
<code>gxPaperTypeNotFound</code>	The default paper-type object cannot be located.

### SEE ALSO

Listing 3-9 on page 3-42 shows how to use the `GXNewFormat` function to create a format object.

To dispose of a format object, see the description of the `GXDisposeFormat` function in the next section.

## GXDisposeFormat

---

You can use the `GXDisposeFormat` function to dispose of a format object.

```
void GXDisposeFormat (gxFormat aFormat);
```

`aFormat`                    A reference to the format object whose owner count you want to decrement.

### DESCRIPTION

You use the `GXDisposeFormat` function when you no longer need the format object. The function decrements the format's owner count. When the owner count reaches 0, the format object is deleted.

**SPECIAL CONSIDERATIONS**

You should *not* call this function for the default format object unless you have cloned it.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

Listing 3-9 on page 3-42 shows how to use the `GXDisposeFormat` function to dispose of a format object.

## **GXCopyFormat**

---

You can use the `GXCopyFormat` function to create a copy of a format object.

```
gxFormat GXCopyFormat (gxFormat srcFormat, gxFormat dstFormat);
```

`srcFormat`    A reference to the source format object to copy.

`dstFormat`    A reference to the destination format object.

*function result*    A reference to a format object.

**DESCRIPTION**

The `GXCopyFormat` function copies the properties from the source format object into the destination object and returns a reference to the destination format object. If you specify `nil` for the `dstFormat` parameter, QuickDraw GX creates a format object to receive the properties.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

**SEE ALSO**

Listing 3-14 on page 3-56 shows how to use the `GXCopyFormat` function to copy a format object's storage.

## GXCloneFormat

---

You can use the `GXCloneFormat` function to increment the owner count of a format object by 1.

```
gxFormat GXCloneFormat (gxFormat aFormat);
```

`aFormat`      A reference to the format object you wish to clone.

*function result*   A reference to a format object.

### DESCRIPTION

When a user wants to share a format with another page in the same document, you use the `GXCloneFormat` function to increment the owner count of a format object by 1, which prevents it from being deleted if it is disposed of when the other page no longer needs the format.

You can use the `GXCountFormatOwners` function to obtain the current owner count of a format object.

### RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

Listing 3-10 on page 3-46 shows how to use the `GXCloneFormat` function to increment the owner count of a format object by 1.

The `GXNewFormat` function, which creates a new format object with an owner count of 1, is described on page 3-104.

The `GXDisposeFormat` function for decrementing the owner count of a format object by 1 is described on page 3-104.

The `GXCountFormatOwners` function for obtaining the current owner count of a format object is described on page 3-107.



## GXCountJobFormats

---

You can use the `GXCountJobFormats` function to obtain the number of format objects in a job object.

```
long GXCountJobFormats (gxJob aJob);
```

`aJob`            A reference to the job object in which to count the format objects.

*function result*   The number of format objects for the job object.

### DESCRIPTION

The `GXCountJobFormats` function determines the number of format objects associated with a particular job object and returns 1 if the default format is the only format object associated with a job object. A job object may contain any number of format objects.

### RESULT CODES

`gxSegmentLoadFailedErr`    A required code segment could not be found, or there was not enough memory to load it.

## GXCountFormatOwners

---

You can use the `GXCountFormatOwners` function to determine the owner count of a format object.

```
long GXCountFormatOwners (gxFormat aFormat);
```

`aFormat`            A reference to the format object in which to obtain the owner count.

*function result*   The owner count.

### DESCRIPTION

The `GXCountFormatOwners` function returns the current number of references to the format object specified by the `aFormat` parameter. The `GXNewFormat` function sets the owner count to 1. The `GXCloneFormat` function increments the owner count of a format object by 1, and the `GXDisposeFormat` function decrements the owner count by 1. When the owner count reaches 0, QuickDraw GX disposes of the format object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

The `GXNewFormat` function, which creates a new format object with an owner count of 1, is described on page 3-104.

The `GXCloneFormat` function, which increments the owner count of a format object by 1, is described on page 3-106.

The `GXDisposeFormat` function for decrementing the owner count of a format object by 1 is described on page 3-104.

## **GXForEachJobFormatDo**

---

You can use the `GXForEachJobFormatDo` function to manipulate each format object in a particular job object.

```
void GXForEachJobFormatDo (gxJob aJob, gxFormatProc aFormatProc,
                          void *refCon);
```

<code>aJob</code>	A reference to the job object associated with a particular format object.
-------------------	---

<code>aFormatProc</code>	A pointer to the function to call for each format object in a job object.
--------------------------	---

<code>refCon</code>	The reference constant passed to the function.
---------------------	--

**DESCRIPTION**

The `GXForEachJobFormatDo` function calls the application-defined function specified in the `aFormatProc` parameter for each format object associated with the job object specified in the `aJob` parameter. The `GXForEachJobFormatDo` function terminates when the application-defined function returns `gxStopLooping` or all format objects associated with the job object have been processed. The first format object to be processed is the default format.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

Listing 3-17 on page 3-60 shows how to use the `GXForEachJobFormatDo` function to access a format object function for each format object in a particular job object.

For information about setting up the function that is called each time through the loop, see “Looping Through Format Objects” on page 3-126.

## Manipulating Format Object Properties

---

You use the `GXGetFormatMapping` function to obtain a format object’s mapping.

You use the `GXGetFormatPaperType` function to obtain a format object’s paper-type object.

To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. To associate a form and its mask shape with a format object, you use the `GXSetFormatForm` function.

You call the `GXChangedFormat` function each time you change a format collection associated with a format object.

## GXGetFormatMapping

---

You can use the `GXGetFormatMapping` function to obtain the mapping for a format object.

```
void GXGetFormatMapping (gxFormat aFormat, gxMapping *aMapping);
```

<code>aFormat</code>	A reference to the format object for which to obtain the mapping.
----------------------	---

<code>aMapping</code>	On return, the mapping for a format object.
-----------------------	---

*function result* None.

**DESCRIPTION**

The `GXGetFormatMapping` function returns a mapping for a format object that is a mathematical representation of the format object’s scaling and orientation settings.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

Listing 3-15 on page 3-57 shows how to use the `GXGetFormatMapping` function.

**GXGetFormatPaperType**

---

You can use the `GXGetFormatPaperType` function to obtain the paper-type object referenced by a format object.

```
gxPaperType GXGetFormatPaperType (gxFormat aFormat);
```

`aFormat`      A reference to the format object for which to obtain the paper-type object.

*function result*   A reference to a paper-type object.

**DESCRIPTION**

The `GXGetFormatPaperType` function returns a reference to a paper-type object as its function result.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

**SEE ALSO**

Listing 3-16 on page 3-59 shows an example that uses the `GXGetFormatPaperType` function.

## GXGetFormatForm

---

You can use the `GXGetFormatForm` function to retrieve the form and mask shapes for a particular format object.

```
gxShape GXGetFormatForm (gxFormat aFormat, gxShape *mask);
```

`aFormat`      A reference to the format object associated with the form and mask shapes.

`mask`            On return, the mask assigned to a format object.

*function result*   A shape that represents the form.

### DESCRIPTION

To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function. To replace any form previously associated with a particular format object, you use the `GXSetFormatForm` function. Picture shapes used by the form are flattened to disk with the format object during spooling.

### RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

To associate a form with a format object, see the description of the `GXSetFormatForm` function in the next section.

## GXSetFormatForm

---

You can use the `GXSetFormatForm` function to associate the form and mask shapes with a specific format object.

```
void GXSetFormatForm (gxFormat aFormat, gxShape form,
                      gxShape mask);
```

`aFormat`      A reference to the format object in which to associate the form and mask shapes.

`form`            A reference to a picture shape that specifies the form to assign to a format object.

**mask**                    A reference to a picture shape that specifies the mask to assign to a format object.

**DESCRIPTION**

The `GXSetFormatForm` function replaces any form previously associated with a particular format object. It increments the owner counts of the new picture shapes (by calling the `GXCloneShape` function) and decrements the owner counts of the old picture shapes (by calling the `GXDisposeShape` function).

You may set either the `form` parameter or the `mask` parameter to `nil`.

Picture shapes are flattened to disk with the format object during spooling. To retrieve the form and mask shapes for a particular format object, you use the `GXGetFormatForm` function.

**RESULT CODES**

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

Listing 3-12 on page 3-51 shows how to use the `GXSetFormatForm` function to associate the form and mask shapes with a specific format object.

To obtain the form shape associated with a format object, see the description of the `GXGetFormatForm` function in the previous section.

## **GXChangedFormat**

---

You can use the `GXChangedFormat` function each time you change a format without directly calling `QuickDraw GX`.

```
void GXChangedFormat (gxFormat aFormat);
```

**aFormat**                    A reference to the format object which you are changing.

**DESCRIPTION**

You need to call the `GXChangedFormat` function each time you change a format object indirectly. For example, you should call this function when you modify a format collection.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**Displaying the Custom Page Setup Dialog Box**

---

To allow a user to change a format object's settings, you need to display the Custom Page Setup dialog box. You use the `GXFormatDialog` function to display the Custom Page Setup dialog box on the user's screen.

**GXFormatDialog**

---

You can use the `GXFormatDialog` function to display the Custom Page Setup dialog box when the user chooses the Custom Page Setup menu item from the File menu.

```
gxDialogResult GXFormatDialog (gxFormat aFormat,
                               gxEditMenuRecord *anEditMenuRecord,
                               StringPtr title);
```

<code>aFormat</code>	A reference to the format object that specifies the values to display in the dialog box.
<code>title</code>	The title of the dialog box.
<code>anEditMenuRecord</code>	A structure for your application's Edit menu and its menu items.

*function result* The user's response to the dialog box.

**DESCRIPTION**

After you use the `GXFormatDialog` function to display the Custom Page Setup dialog box, the user can specify formatting information for a format (which is not the default format). For example, the user can specify the paper size, orientation, and the default formatting printer.

In the `anEditMenuRecord` parameter you to specify an Edit menu structure to support the standard editing operations of cut, copy, paste, and clear in dialog boxes.

The `GXFormatDialog` function returns a response that is defined in a dialog box result enumeration. If the user chooses the Format button, the `GXFormatDialog` function returns `gxOKSelected`. If the user chooses the Cancel button, the function returns `gxCancelSelected`. If the user chooses the Remove button, the function returns `gxRevertSelected`.

If an error occurs, the function returns `gxCancelSelected`. Call the `GXGetJobError` function to determine which error occurred.

This function causes QuickDraw GX to send the `gxFormatDialog` message, which you can override to customize the Custom Page Setup dialog box.

Note that QuickDraw GX stores a user's responses to some dialog box items in the Custom Page Setup dialog box in a format collection.

#### SEE ALSO

Listing 3-9 on page 3-42 shows how to use the `GXFormatDialog` function to display the Custom Page Setup dialog box.

The Edit menu structure and the dialog box result enumeration are described in the chapter “Core Printing Features” in this book.

For information about customizing the Custom Page Setup dialog box, see “Adding Panels to Dialog Boxes” beginning on page 3-67.

## Working With Panels

---

The following functions allow you to add panels to a dialog box. The `GXSetupDialogPanel` function adds a panel to a dialog box.

You use the `GXGetJobPanelDimensions` function to obtain the dimensions of a panel. This function allows you to locate the position of the cursor within a dialog panel.

You use the `GXEnableJobScalingPanel` function to prevent the display of the default scaling field in the Page Setup and Custom Page Setup dialog boxes. For example, if you implement your own scaling panel, you would disable the default scaling field provided by QuickDraw GX.

You typically call these methods from within an override function for the message that displays the panel. See the section “Message Override Functions for Customizing QuickDraw GX Dialog Boxes” beginning on page 3-119 for information about these message override functions.

## GXSetupDialogPanel

---

You can use the `GXSetupDialogPanel` function to add a panel to a print dialog box.

```
OSErr GXSetupDialogPanel (gxPanelSetupRecord *panelRec);
```

`panelRec`     A pointer to a panel setup structure.

*function result*     An error code. The value `noErr` indicates that the operation was successful.



**DESCRIPTION**

The `GXSetupDialogPanel` function adds a panel, as defined by the information in the panel setup structure, to a print dialog box. You call this function from within your override of the `gxJobPrintDialog`, `gxFormatDialog`, and `gxJobDefaultFormatDialog` messages, before forwarding the message.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.
<code>gxCantAddPanelsNowErr</code>	Panels can only be added to a dialog box when the current driver is switched. This error is generated if a panel addition request is made at any other time.
<code>gxBadxdtlKeyErr</code>	An unknown key value was specified for an item in an extended dialog control resource.
<code>gxXdtlItemOutOfRangeErr</code>	An item referenced by the panel does not belong to the panel.
<code>gxNoActionButtonErr</code>	The action button for the panel is <code>nil</code> .
<code>gxTitlesTooLongErr</code>	The length of the button titles exceeds the maximum width allowed for a printing alert.

**SEE ALSO**

Listing 3-22 on page 3-68 shows how to use the `GXSetupDialogPanel` function to add a panel to the Custom Page Setup dialog box.

## **GXGetJobPanelDimensions**

---

You can use the `GXGetJobPanelDimensions` function to obtain the dimensions of the area for the panel within a dialog box.

```
void GXGetJobPanelDimensions (gxJob aJob, Rect *aRect);
```

<code>aJob</code>	A reference to the job object associated with the panel.
<code>aRect</code>	On return, the rectangle whose geometry specifies the panel's size.

**DESCRIPTION**

When you want to locate the position of the cursor within a panel, you use the `GXGetJobPanelDimensions` function to obtain the dimensions of a panel.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**GXEnableJobScalingPanel**

---

You can use the `GXEnableJobScalingPanel` function to prevent display of the default QuickDraw GX scaling field in the Page Setup and Custom Page Setup dialog boxes.

```
void GXEnableJobScalingPanel (gxJob aJob, Boolean enabled);
```

<code>aJob</code>	A reference to the job object associated with the scaling field.
<code>enabled</code>	A Boolean value that specifies whether or not to enable the scaling field.

**DESCRIPTION**

The `GXEnableJobScalingPanel` function enables or disables the scaling field. You set the `enabled` parameter to `true` to enable the scaling field and `false` to disable it. For example, you might disable this field if you want to provide your own scaling panel instead of the default field. The scaling field is enabled by default.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**GXGetMessageHandlerResFile**

---

You can use the `GXGetMessageHandlerResFile` function to retrieve the resource file reference number of the printing extension or printer driver.

```
short GXGetMessageHandlerResFile (void);
```

<i>function result</i>	The resource file reference number of the printing extension or printer driver.
------------------------	---

**DESCRIPTION**

The `GXGetMessageHandlerResFile` function returns the resource file reference number for the printing extension or printer driver. You can use this function if you need to access data from these resources.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

**Accessing Printing-Related Collection Objects**

---

When you want to obtain a collection object associated with a particular job object, format object, or paper-type object, you use the `GXGetJobCollection`, `GXGetFormatCollection`, and `GXGetPaperTypeCollection` functions. For more information about collections, see the Collection Manager chapter of *Inside Macintosh: Environment and Utilities*.

**GXGetJobCollection**

---

You can use the `GXGetJobCollection` function to obtain the job collection object associated with a particular job object.

```
Collection GXGetJobCollection (gxJob aJob);
```

`aJob`            A reference to the job object whose collection object you want to obtain.

*function result*   A reference to a collection object.

**DESCRIPTION**

After you call the `GXGetJobCollection` function to obtain a job collection object, you must call the `GXGetJobError` function to obtain errors. It is important that you resolve errors immediately because the Collection Manager cannot work with a `nil` collection object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

Listing 3-3 on page 3-28 shows how to use the `GXGetJobCollection` function to obtain the collection object associated with a particular job object.

## GXGetFormatCollection

---

You can use the `GXGetFormatCollection` function to obtain the format collection object associated with a particular format object.

```
Collection GXGetFormatCollection (gxFormat aFormat);
```

`aFormat`      A reference to the format object whose collection object you want to obtain.

*function result*   A reference to a collection object.

### DESCRIPTION

After you call the `GXGetFormatCollection` function to obtain a format collection object, you must call the `GXGetJobError` function to obtain errors. It is important that you resolve errors immediately because the Collection Manager cannot work with a `nil` collection object.

### RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

## GXGetPaperTypeCollection

---

You can use the `GXGetPaperTypeCollection` function to obtain the paper-type collection object associated with a particular paper-type object.

```
Collection GXGetPaperTypeCollection (gxPaperType aPaperType);
```

`aPaperType`      A reference to the paper-type object whose collection object you want to obtain.

*function result*   A reference to a collection object.

### DESCRIPTION

After you call the `GXGetPaperTypeCollection` function to obtain a paper-type collection object, you must call the `GXGetJobError` function to obtain errors. It is important that you resolve errors immediately because the Collection Manager cannot work with a `nil` collection object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## Application-Defined Functions

---

The following sections describe the functions that you must provide if you want to override QuickDraw GX print dialog messages or manipulate the format objects associated with a job object.

### Message Override Functions for Customizing QuickDraw GX Dialog Boxes

---

To install an override function for a message, you need to call the `GXInstallApplicationOverride` function. Within the `GXInstallApplicationOverride` function, you specify a pointer to a function that you use to override a particular message for a specific dialog box. These messages include

- n `gxJobPrintDialog`
- n `gxJobDefaultFormatDialog`
- n `gxFormatDialog`
- n `gxHandlePanelEvent`
- n `gxFilterPanelEvent`
- n `gxParsePageRange`

You can use dialog box messages to invoke your actions. QuickDraw GX sends the `gxJobDefaultFormatDialog` message when your application calls `GXJobDefaultFormatDialog` to display the Page Setup dialog box. QuickDraw GX sends the `gxFormatDialog` message when your application calls `GXFormatDialog` to display the Custom Page Setup dialog box and it sends the `gxJobPrintDialog` message when your application calls `GXJobPrintDialog` to display the Print dialog box.

QuickDraw GX also sends the `gxHandlePanelEvent` message and the `gxFilterPanelEvent` message when an event occurs in a panel.

## GXJobPrintDialog

---

QuickDraw GX sends the `gxJobPrintDialog` message when the application calls `gxJobPrintDialog` to display the Print dialog box. You can install an override function for the `gxJobPrintDialog` message to modify the behavior or appearance of the Print dialog box. Your override function must match the following formal declaration:

```
OSErr GXJobPrintDialog (gxDialogResult *aDialogResult);
```

`aDialogResult`

On return, a pointer to the selection made by the user in the dialog box.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxJobPrintDialog` message when the user selects Print from the File menu and the application subsequently calls the `GXJobPrintDialog` function to display the Print dialog box on the user's screen.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function.

### SPECIAL CONSIDERATIONS

You never send the `gxJobPrintDialog` message yourself.

You must forward the `gxJobPrintDialog` message to other message handlers. Add your panels and then forward the message.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The user has canceled printing.

## GXJobDefaultFormatDialog

---

QuickDraw GX sends the `gxJobDefaultFormatDialog` message when the application calls the `GXJobDefaultFormatDialog` function to display the Page Setup dialog box. You can install an override function for the `gxJobDefaultFormatDialog` message to modify the behavior or appearance of the dialog box. Your override function must match the following formal declaration:

```
OSErr GXJobDefaultFormatDialog (gxDialogResult *aDialogResult);
```

`aDialogResult`

On return, a pointer to a value that specifies the selection made by the user in the dialog box.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxJobDefaultFormatDialog` message when the user clicks the More Choices button in the Page Setup dialog box. The application calls the `GXJobDefaultFormatDialog` function to display the extended Page Setup dialog box.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function. You can add your own panels to the dialog box through the normal QuickDraw GX printing calls.

### SPECIAL CONSIDERATIONS

You never send the `gxJobDefaultFormatDialog` message yourself.

You must forward the `gxJobDefaultFormatDialog` message to other message handlers. Add your panels and then forward the message.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

## GXFormatDialog

---

QuickDraw GX sends the `gxFormatDialog` message when the application calls the `GXFormatDialog` function to display the Custom Page Setup dialog box. You can install an override function for the `gxFormatDialog` message to modify the behavior or appearance of the dialog box. Your override function must match the following formal declaration:

```
OSErr GXFormatDialog (gxFormat aFormat, StringPtr title,
                     gxDialogResult *aDialogResult);
```

<code>aFormat</code>	A reference to the format object.
<code>title</code>	The title of the dialog box. If you specify <code>nil</code> as the value of this parameter, the title “Custom Page Setup” is used.
<code>aDialogResult</code>	On return, a pointer to the selection made by the user in the dialog box.
<i>function result</i>	An error code. The value <code>noErr</code> indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxFormatDialog` message when the user selects the Custom Page Setup menu item and an application subsequently calls the `GXFormatDialog` function to display the Custom Page Setup dialog box.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function.

### SPECIAL CONSIDERATIONS

You never send the `gxFormatDialog` message yourself.

You must forward the `gxFormatDialog` message to other message handlers. Add your panels and then forward the message.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.



## GXHandlePanelEvent

---

QuickDraw GX sends the `gxHandlePanelEvent` message when an event happens in a panel. You can install an override function for the `gxHandlePanelEvent` message to handle panel events that cannot be handled using extended item list resources. Your override function must match the following formal declaration:

```
OSErr GXHandlePanelEvent (gxPanelInfoRecord *aPanelInfoRecord,
                          gxPanelResult *panelResult);
```

`aPanelInfoRecord`

A pointer to the panel information structure that supplies information to the panel about the current dialog box and panel event.

`panelResult`

On return, the result of handling the panel event.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxHandlePanelEvent` message to allow a panel to handle events associated with the dialog box. The result code returned by the `panelResult` parameter is either a value of type `OSErr`, or one of the following values:

`gxPanelNoResult`

The returned value does not currently have any meaning.

`gxPanelCancelConfirmation`

The user confirmed the panel, however, the panel handler discovers that the user entered an inappropriate value in the dialog box.

The default implementation of this message does nothing. You need to override this message if you add panels that cannot be handled using extended item list resources.

### SPECIAL CONSIDERATIONS

You never send the `gxHandlePanelEvent` message yourself.

You always perform a total override of the `gxHandlePanelEvent` message, in which you handle any events of interest that occur in your panel.

### RESULT CODES

`gxSegmentLoadFailedErr`

A required code segment could not be found, or there was not enough memory to load it.

`gxPrUserAbortErr`

The user has canceled printing.

## GXFilterPanelEvent

---

QuickDraw GX sends the `gxFilterPanelEvent` message when an event happens in a panel. You can install an override function for the `gxFilterPanelEvent` message to add panels that need a filter procedure. Your override function must match the following formal declaration:

```
OSErr GXFilterPanelEvent (gxPanelInfoRecord *aPanelInfoRecord;
                          Boolean *returnImmed);
```

`aPanelInfoRecord`

A pointer to the panel information structure that supplies information to the panel about the current dialog box and panel event.

`returnImmed`

On return, a Boolean value that is `true` if there should be no further processing on this event and `false` if not.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxFilterPanelEvent` message to filter panel events in a dialog box.

The default implementation of this message does nothing. You need to override this message if you add panels that require a filtering process.

### SPECIAL CONSIDERATIONS

You never send the `gxFilterPanelEvent` message yourself.

You always perform a total override of the `gxFilterPanelEvent` message, in which you filter any events that occur in your panel.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

## GXParsePageRange

---

QuickDraw GX sends the `gxParsePageRange` message when the user selects a range of pages for printing. You can install an override function for the `gxParsePageRange` message to modify or validate the page range. Your override function must match the following formal declaration:

```
OSErr GXParsePageRange (StringPtr fromString, StringPtr toString,
                        gxParsePageRangeResult *result);
```

`fromString` A pointer to the string representation of the From page.

`toString` A pointer to the string representation of the To page.

`result` On return, a value that specifies the result code for the range parsing. The constants for this value are given in the section “The Panel Setup Structure” on page 3-101.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxParsePageRange` message to validate that a page range entered by the user is appropriate for the print job.

The default implementation of this message adds the standard printing panels and interface and then displays the dialog box.

You usually override this message to customize the dialog box by adding panels using the `GXSetupDialogPanel` function.

### SPECIAL CONSIDERATIONS

You rarely need to send the `gxParsePageRange` message yourself.

You must forward the `gxParsePageRange` message to other message handlers.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

## Looping Through Format Objects

---

When you want to make changes to each format object associated with a document, you can use the `GXForEachJobFormatDo` function to access format objects. In this function you must provide a pointer to a format function.

To access each format object associated with a printable document, provide a pointer to a format function in the `GXForEachJobFormatDo` function that takes two parameters: the format object associated with a particular job object, and a pointer to a reference constant in which you specify unique format object references. For example, this is how you should declare the function if you were to name it `MyFormatFunction`:

```
gxLoopStatus MyFormatFunction (gxFormat aFormat, void *refCon);
```

`aFormat`        The current format. This is provided by QuickDraw GX when the function is called.

`refCon`        A pointer to a reference constant for each format object.

*function result*   A Boolean value to indicate whether looping should stop.

### DESCRIPTION

When you use the `GXForEachJobFormatDo` function, QuickDraw GX calls your format function multiple times as it retrieves each format object referenced by a job object.

## Dialog Box-Related Resources

---

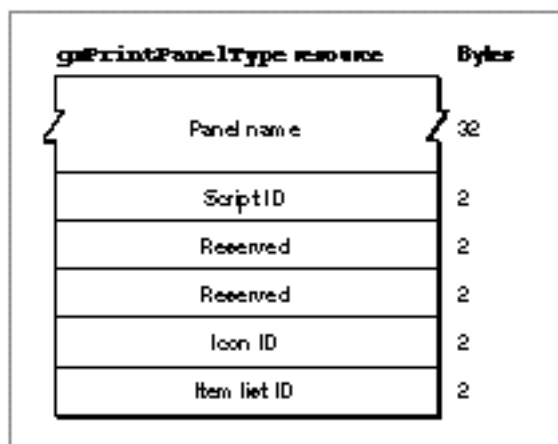
This section describes the resources that you use when you add panels to QuickDraw GX dialog boxes.

### The Panel Resource

---

Figure 3-20 shows the format of the compiled panel resource, `gxPrintPanelType`.

**Figure 3-20** Panel resource



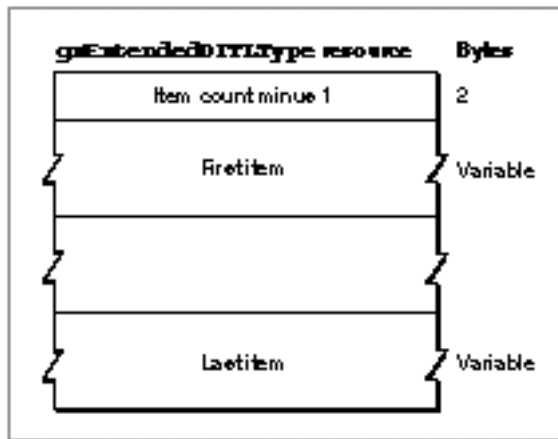
The compiled version of a panel resource contains the following elements:

- n Panel name. This is a Pascal string that contains the name of the panel.
- n Script ID. This is the name of the script in which the panel is written; for example, `smRoman`.
- n Reserved. These words are reserved for future use.
- n Icon ID. This is the resource ID for the icon resource that displays in the expanded dialog box.
- n Item ID. This is the resource ID of the items that are displayed in the panel.

## The Extended Item List Resource

Figure 3-21 shows the format of the compiled extended item resource, `gxExtendedDITLType`.

**Figure 3-21** Extended item list resource



The compiled version of this resource contains the following elements:

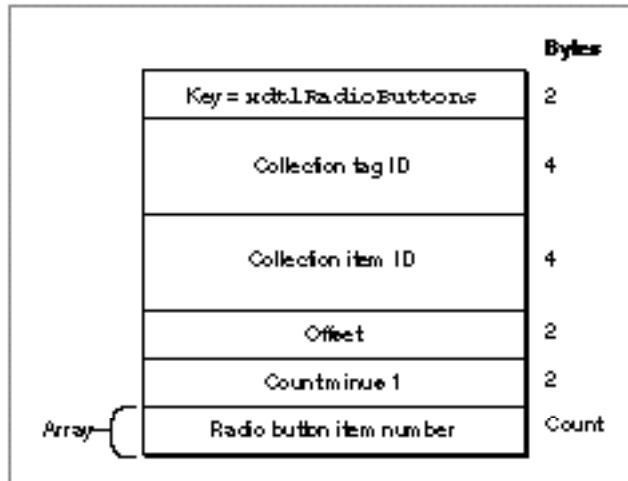
- n Item count - 1. This is the number of items in the resource, less 1.
- n A variable number of items.

The format of each item depends on its type, as defined below.

<code>xdtlRadioButtons</code>	Radio buttons
<code>xdtlCheckBox</code>	Checkbox
<code>xdtlEditTextInteger</code>	Integer-format editable text
<code>xdtlEditTextReal</code>	Real-format editable text
<code>xdtlEditTextString</code>	String-format editable text
<code>xdtlPopup</code>	Pop-up menu

The compiled version of an item for a group of radio buttons is shown in Figure 3-22.

**Figure 3-22** Radio button items

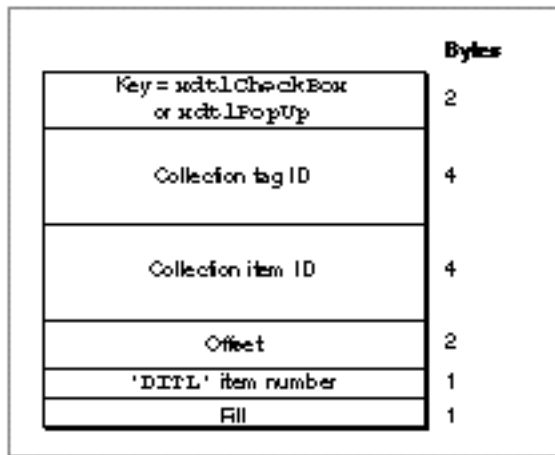


It contains the following elements:

- n **Key.** The key specifies the kind of item. It is always `xdtl.RadioButton`.
- n **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- n **Collection item ID.** The item ID specifies the collection item ID, such as `'incf'` for the level of fonts to include.
- n **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- n **Count.** The count specifies the number of radio buttons in this list. Because there is 1 byte per button in the collection item, the count also specifies the size for the group of buttons in the collection item.
- n **Item number.** The item number specifies the item list's item that corresponds to this radio button. There is one item per button.

The compiled version of an item for both a checkbox or pop-up menu is shown in Figure 3-23.

**Figure 3-23** Checkbox and pop-up menu items

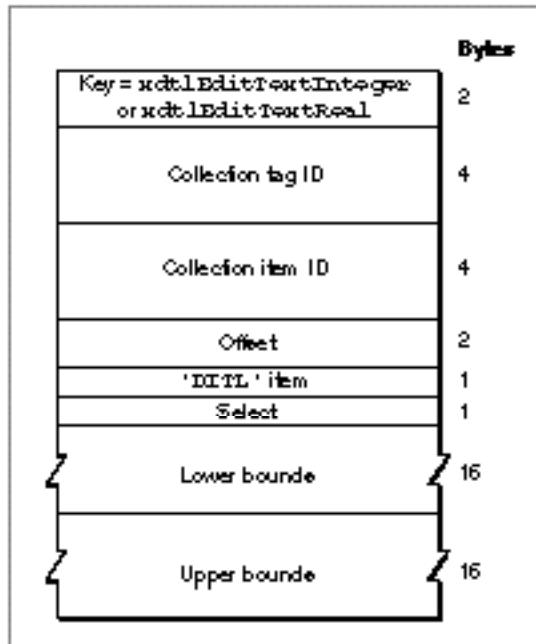


It contains the following elements:

- n **Key.** The key specifies the kind of item. It is always `xdtlCheckBox` for checkboxes and `xdtlPopUp` for pop-up menus.
- n **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- n **Collection item ID.** The item ID specifies the collection item ID, such as `'dest'` for whether to print to a file.
- n **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- n **Item number.** The item number specifies the item list's item that corresponds to this checkbox or pop-up menu.
- n **Fill.** The fill is 1 byte.

The compiled version of an item for both integer or real editable text is shown in Figure 3-24.



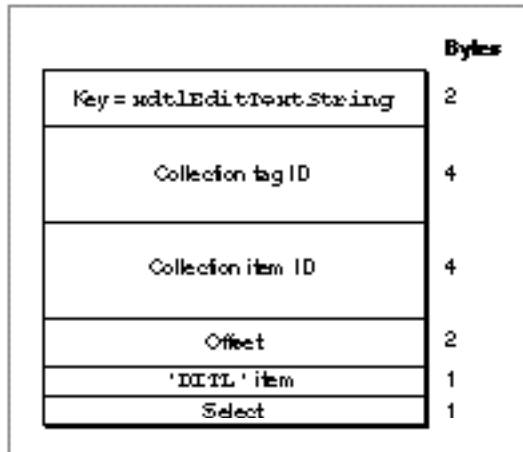
**Figure 3-24** Integer and real edit text items

It contains the following elements:

- n **Key.** The key specifies the kind of item. It is always `xdtlEditTextInteger` for integers and `xdtlEditTextReal` for real numbers.
- n **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- n **Collection item ID.** The item ID specifies the collection item ID, such as `'copy'` for the number of copies.
- n **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- n **Item number.** The item number specifies the item list's item that corresponds to the editable text item.
- n **Select.** This element specifies whether or not to highlight the text. If its value is 0, the text is not highlighted. If its value is 1, the text is highlighted.
- n **Lower bounds.** This is a Pascal string that contains an optional sign (plus or minus), digits, and for real numbers, an optional decimal point before the fractional part of the number. If the string is `nil`, no lower bounds is specified.
- n **Upper bounds.** This is a Pascal string that contains an optional sign (plus or minus), digits, and for real numbers, an optional decimal point before the fractional part of the number. If the string is `nil`, no upper bounds is specified.

The compiled version of an item for string editable text is shown in Figure 3-25.

**Figure 3-25** String editable text items



It contains the following elements:

- n **Key.** The key specifies the kind of item. It is always `xdtlEditTextString`.
- n **Collection tag ID.** The collection tag specifies the creator of the collection item, such as `gxPrintingTagID` for items provided by QuickDraw GX in the job, format, and paper-type collections.
- n **Item ID.** The item ID specifies the collection item ID, such as `'incf'` for the level of fonts to include.
- n **Offset.** The offset specifies the start of storage for the data. It is the number of bytes into the collection item.
- n **Item number.** The item number specifies the item list's item that corresponds to the editable text item.
- n **Select.** This element specifies whether or not to highlight the text. If its value is 0, the text is not highlighted. If its value is 1, the text is highlighted.

## Summary of Page Formatting and Dialog Box Customization

---

### Constants and Data Types

---

#### Constants for Loop Status Information

---

```
enum {
    gxStopLooping = false, /* stop looping */
    gxKeepLooping = true   /* keep looping */
};

typedef Boolean gxLoopStatus;

/* function for each format object associated with a job object */
typedef gxLoopStatus (*gxFormatProc) (gxFormat aFormat, void *refCon);
```

#### Constants for Collection Item Categories and Tag IDs

---

##### Collection Item Categories

```
typedef short gxCollectionCategory; /* stored in collection object items' */
/* user attribute bits */

enum {
    gxNoCollectionCategory= (gxCollectionCategory) 0x0000, /* not volatile */
    gxOutputDriverCategory= (gxCollectionCategory) 0x0001, /* affected by out-
                                                             put printer */
    gxFormattingDriverCategory= (gxCollectionCategory) 0x0002, /* affected by
                                                             formatting
                                                             printer */
    gxDriverVolatileCategory= (gxCollectionCategory) 0x0004, /* volatile */

    gxVolatileOutputDriverCategory =
        /* purge when output printer driver changes */
        gxOutputDriverCategory + gxDriverVolatileCategory,
```

```

gxVolatileFormattingDriverCategory =
    /* purge when formatting printer driver changes */
    gxFormattingDriverCategory + gxDriverVolatileCategory
};

```

### Collection Tag ID

```

enum { gxPrintingTagID = -28672 }; /* QuickDraw GX assigns its collection
                                   objects with the same 4-byte ID */

```

### Constants and Data Types for Job Collection Items

---

#### Print-Job Information

```

enum { gxJobTag = 'job ' }; /* item ID for the print-job item */

/* job object information structure */
struct gxJobInfo {
    long    numPages;          /* total number of pages to print */
    long    priority;          /* print job's priority */
    long    timeToPrint;       /* designated time to print a print job */
    long    jobTimeout;        /* time to cancel print job, in ticks */
    long    firstPageToPrint   /* first page to begin printing */
    short   jobAlert;          /* when to alert the user about printing */
    Str31   appName;           /* name of application used to create the */
                                /* printable document */
    Str31   documentName;      /* name of the user's document */
    Str31   userName;          /* name of the user associated with the */
                                /* printable document */
};

enum {
    /* print-job priorities */
    gxPrintJobUrgent = 0x00000001, /* priority of print job is */
                                    /* "urgent" */
    gxPrintJobAtTime = 0x00000002, /* designated time to print the */
                                    /* print job */
    gxPrintJobASAP    = 0x00000003 /* designated time to print the */
                                    /* print job is "as soon as */
                                    /* possible" */
};

```

```

enum { gxPrintJobHoldingBit = 0x00001000 }; /* reserved bit in the
                                              priority field indicates a
                                              print job on hold */

enum {
    /* print-job holding status */
    gxPrintJobHolding      = (gxPrintJobHoldingBit + gxPrintJobASAP),
    gxPrintJobHoldingAtTime = (gxPrintJobHoldingBit + gxPrintJobAtTime),
    gxPrintJobHoldingUrgent = (gxPrintJobHoldingBit + gxPrintJobUrgent)
};

enum {
    /* print-job alert constants */
    gxNoPrintTimeAlert= 0, /* don't alert user when printing */
    gxAlertBefore      = 1, /* alert user before printing */
    gxAlertAfter       = 2, /* alert user after printing */
    gxAlertBothTimes   = 3 /* alert user before and after printing */
};

enum {
    /* time to cancel print job */
    gxThirtySeconds    = 1800, /* cancel print job in 30 seconds (in ticks) */
    gxTwoMinutes       = 7200 /* cancel print job in 2 minutes (in ticks) */
};

```

### Collation Information

```

enum { gxCollationTag = 'sort' }; /* item ID for the collation item*/

/* collation information structure */
struct gxCollationInfo {
    Boolean collation; /* indicates whether or not to collate */
                      /* copies */
};

```

### Copies Information

```

enum { gxCopiesTag = 'copy' }; /* item ID for the copies item*/

/* copies information structure */
struct gxCopiesInfo {
    long copies; /* number of copies of a document to print */
};

```

**Page-Range Information**

```

enum { gxPageRangeTag = 'rang' }; /* item ID for the page-range item */

/* page-range information structure */
struct gxPageRangeInfo {
    gxSimplePageRangeInfo    simpleRange;    /* simple page range */
                                           /* information */
    Str31                    fromString;      /* beginning of customized */
                                           /* page range */
    Str31                    toString;        /* end of customized page */
                                           /* range */
    long                     minFromPage;     /* minimum of default page */
                                           /* range */
    long                     maxToPage;       /* maximum of default page */
                                           /* range */
    char                     replaceString[1]; /* page-range replacement */
                                           /* string */
} ;

/* simple page-range information structure */
struct gxSimplePageRangeInfo {
    char    optionChosen; /* specific page-range option */
    Boolean printAll;     /* true if user wants to print all pages of a */
                        /* document */
    long    fromPage;     /* first page in page range */
    long    toPage;       /* last page in page range */
};

enum {
    /* page-range options */
    gxDefaultPageRange = (char) 0, /* use default numeric page range */
    gxReplacePageRange = (char) 1, /* use editable text field */
    gxCustomizePageRange = (char) 2 /* use alphanumeric page range */
};

```

**Quality Information**

```
enum { gxQualityTag = 'qual' }; /* item ID for the quality item*/

/* quality information structure */
struct gxQualityInfo {
    Boolean    disableQuality; /* true to disable standard quality */
                                /* controls */
    short      defaultQuality; /* default quality */
    short      currentQuality; /* current quality */
    short      qualityCount;   /* number of quality menu items in */
                                /* Quality pop-up menu */
    char       qualityNames[1]; /* Quality pop-up menu names, such as */
                                /* "Best" */
};
```

**File-Destination Information**

```
enum { gxFileDestinationTag = 'dest' }; /* item ID for the file- */
                                         /* destination item*/

/* file-destination information structure */
struct gxFileDestinationInfo {
    Boolean toFile; /* true if destination is a file */
};
```

**File-Location Information**

```
enum { gxFileLocationTag = 'floc' } /* item ID for the file- */
                                     /* location item*/

/* file-location information structure */
struct gxFileLocationInfo {
    FSSpec fileSpec; /* location of file */
};
```

**File-Format Information**

```
enum { gxFileFormatTag = 'ffmt' }; /* item ID for the file- */
                                     /* format item*/
```

```

/* file-format information structure */
struct gxFileFormatInfo {
    Str31 fileFormatName;          /* name of file format */
};

```

### File-Fonts Information

```

enum { gxFileFontsTag = 'incf' }; /* item ID for the file-fonts item */

/* file-fonts information structure */
struct gxFileFontsInfo {
    char          includeFonts;    /* font include level; if destination is
                                   file */
};

enum {                                /* font include levels */

    gxIncludeNoFonts          = (char) 1,
    gxIncludeAllFonts         = (char) 2,
    gxIncludeNonStandardFonts = (char) 3
};

```

### Paper-Feed Information

```

enum { gxPaperFeedTag = 'feed' }; /* item ID for paper-feed item */

/* paper-feed information structure */
struct gxPaperFeedInfo {
    Boolean      autoFeed;        /* true if automatic feed, false if */
                                   /* manual feed */
};

```

### Manual-Feed Information

```

enum { gxManualFeedTag = 'manf' }; /* item ID for manual-feed item */

/* manual-feed information structure */
struct gxManualFeedInfo {
    long         numPaperTypeNames; /* number of paper-type objects to
                                   /* manually feed */
    Str31        paperTypeNames[1]; /* names of paper-type objects to */
                                   /* manually feed */
};

```



**Standard Mapping Information**

```
enum { gxNormalMappingTag = 'nmap' }; /* item ID for the standard */
                                     /* mapping item */

/* standard mapping information structure */
struct gxNormalMappingInfo {
    Boolean      normalPaperMapping; /* true if not overriding standard */
                                     /* paper matching */
};
```

**Special Mapping Information**

```
enum { gxSpecialMappingTag = 'smap' }; /* item ID for special mapping */

/* special mapping information structure */
struct gxSpecialMappingInfo {
    char          specialMapping;      /* specific paper-mapping option */
} ;

enum {
    /* paper-mapping options */
    gxRedirectPages = (char) 1, /* user wants to crop redirected pages */
    gxScalePages    = (char) 2, /* user wants to scale pages */
    gxTilePages     = (char) 3, /* user wants to tile pages */
};
```

**Tray-Mapping Information**

```
enum { gxTrayMappingTag = 'tmap' };

struct gxTrayMappingInfo{
    gxTrayIndex mapPaperToTray; /* tray to map all paper to */
};

typedef long gxTrayIndex; /* specifies the paper tray setting */
```

**Print-Panel Information**

```
enum { gxPrintPanelTag = 'ppan' }; /* item ID for the Print */
                                     /* panel item */
```

```

/* print-panel information structure */
struct gxPrintPanelInfo {
    Str31    startPanelName;          /* name of starting panel in
                                      /* Print dialog box */
};

```

### Format-Panel Information

```

enum { gxFormatPanelTag = 'fpan' };    /* item ID for the format */
                                      /* panel item */

/* format-panel information structure */
struct gxFormatPanelInfo {              /* name of starting panel in */
    Str31    startPanelName;          /* Page Setup dialog box */
};

```

### Paper-Mapping Information

```

enum { gxPaperMappingTag = 'pmap' }; /* item ID for print- */
                                      /* panel item */

/* This collection item contains a flattened paper-type object resource. */

```

### Translated-Document Information

```

enum { gxTranslatedDocumentTag = 'trns' };

struct gxTranslatedDocumentInfo {
    long translatorInfo;              /* information from the translation process */
};

```

## Constants and Data Types for Format Collection Items

---

### Orientation Information

```

enum { gxOrientationTag = 'layo' };    /* item ID for the */
                                      /* orientation item */

/* orientation information structure */
struct gxOrientationInfo {
    char      orientation;            /* an enumerated orientation value */
} ;

```

```
enum {
    /* orientation options */
    gxPortraitLayout      = (char) 0, /* user wants portrait orientation */
    gxLandscapeLayout     = (char) 1, /* user wants landscape orientation */
    gxRotatedPortraitLayout = (char) 2, /* rotated portrait orientation,
                                         not user specifiable*/
    gxRotatedLandscapeLayout = (char) 3 /* user wants rotated landscape */
                                         /* orientation */
};
```

### Scaling Information

```
enum { gxScalingTag = 'scal' }; /* item ID for the scaling item */

/* scaling information structure */
struct gxScalingInfo {
    Fixed    horizontalScaleFactor; /* current horizontal scaling */
                                         /* factor */
    Fixed    verticalScaleFactor; /* current vertical scaling factor */
    short    minScaling; /* minimum current scaling factor */
    short    maxScaling; /* maximum current scaling factor*/
};
```

### Direct-Mode Information

```
enum { gxDirectModeTag = 'dirm' }; /* item ID for the direct- */
                                         /* mode item */

/* direct-mode information structure */
struct gxDirectModeInfo {
    Boolean    directModeOn; /* true if direct mode is enabled */
};
```

### Format-Halftone Information

```
enum { gxFormatHalftoneTag = 'half' }; /* item ID for the special */
                                         /* mapping item */

/* format-halftone information structure */
struct gxFormatHalftoneInfo {
    long    numHalftones; /* number of halftones */
    gxHalftone    halftones[1]; /* any number of halftones */
};
```

**Page-Inversion Information**

```
enum { gxInvertPageTag = 'invp' }; /* item ID for the page- */
                                   /* inversion item */

/* page-inversion information structure */
struct gxInvertPageInfo {
    Boolean invert;                /* if true, invert the page */
};                                /* if missing or false, don't invert */
                                   /* the page */
```

**Horizontal Page-Flip Information**

```
enum { gxFlipPageHorizontalTag = 'flph' }; /* item ID for the */
                                           /* horizontal page-flip item */

/* horizontal flip-page information structure */
struct gxFlipPageHorizontalInfo{
    Boolean flipHorizontal;        /* if true, flip x coordinates on the */
};                                /* page; if missing or false, don't flip */
```

**Vertical Page-Flip Information**

```
enum { gxFlipPageVerticalTag = 'flpv' }; /* item ID for the */
                                           /* vertical page-flip item */

/* vertical page-flip information structure */
struct gxFlipPageVerticalInfo {
    Boolean flipVertical;          /* if true, flip y coordinates on the */
};                                /* page; if missing or false, don't flip */
```

**Precise-Bitmap Information**

```
enum { gxPreciseBitmapsTag = 'pbmp' }; /* item ID for the precise- */
                                           /* bitmap item */

/* precise-bitmap information structure */
struct gxPreciseBitmapInfo {
    Boolean preciseBitmaps;        /* if true, scale the page by 96% */
};                                /* if missing or false, don't scale */
```

**Paper-Type Lock Information**

```
enum { gxPaperTypeLockTag = 'ptlk' }; /* item ID for the paper- */
                                      /* type lock item*/

/* paper-type object lock information structure */
struct gxPaperTypeLockInfo {
    Boolean    paperTypeLocked;      /* true if paper-type object */
};                                  /* is locked */
```

**Constants and Data Types for Paper-Type Collection Items**

---

**Base Information**

```
enum { gxBaseTag = 'base' };          /* item ID for the base item */

/* base type information structure */
struct gxBaseInfo {
    long    baseType;                /* base type chosen */
} ;

enum {
    /* paper-type object base types */
    gxUnknownBase = 0,              /* unknown base type */
    gxUsLetterBase = 1,             /* US letter base type */
    gxUsLegalBase = 2,             /* US legal base type */
    gxA4LetterBase = 3,            /* A4 letter base type */
    gxB5LetterBase = 4,            /* B5 letter base type */
    gxTabloidBase = 5              /* tabloid base type */
};
```

**Creator Information**

```
enum { gxCreatorTag = 'crea' }; /* item ID for the creator item */

/* creator information structure */
struct gxCreatorInfo {
    OSType    creator;              /* creator of the paper-type object */
};

enum {
    /* paper-type object creator type */
    gxSysPaperType = 'sypt',        /* system paper-type object creator */
};
```

```

gxUserPaperType    = 'uspt'          /* user paper-type object creator */
/* if printer driver creates a paper-type object, use printer
/* driver's creator type */
};

```

### Units Information

```

enum { gxUnitsTag = 'unit' };      /* item ID for the units item */

/* unit information structure */
struct gxUnitsInfo {
    char    units;                  /* specific paper-type object */
                                      /* measurement */
};

enum {
    /* paper-type object units */
    gxPicas  = (char) 0,           /* pica measurement */
    gxMms    = (char) 1,           /* millimeter measurement */
    gxInches = (char) 2            /* inches measurement */
};

```

### Flags Information

```

enum { gxFlagsTag = 'flag' }; /* item ID for the flags item */

/* flags information structure */
typedef struct {
    long flags;                    /* paper-type object flags */
}gxFlagsInfo;

enum {
    /* paper-type object flags (bit positions) */
    gxOldPaperTypeFlag    = 0x00800000, /* indicates a paper-type object */
                                      /* with 7.0 settings */
    gxNewPaperTypeFlag    = 0x00400000, /* indicates a paper-type object */
                                      /* with post 7.0 settings */
    gxOldAndNewPaperTypeFlag= 0x00C00000, /* indicates a paper-type object */
                                      /* that is both old and new */
    gxDefaultPaperTypeFlag = 0x00100000, /* indicates the default paper */
                                      /* type */
};

```

**Comment Information**

```
enum { gxCommentTag = 'cmnt' }; /* item ID for the comment item */

/* comment information structure */
struct gxCommentInfo {
    Str255    comment;          /* paper-type object comment */
} ;
```

**Panel-Related Constants and Data Types**

---

**QuickDraw GX Dialog Box Panel Information**

```
/* constants for overriding messages when adding dialog box panels */
#define gxJobStatus                3
#define gxPrintingEvent            4
#define gxJobDefaultFormatDialog  5
#define gxFormatDialog             6
#define gxJobPrintDialog           7
#define gxFilterPanelEvent         8
#define gxHandlePanelEvent         9
#define gxParsePageRange          10

/* dialog box related resources */
#define gxXdtlRadioButtons         0
#define gxXdtlCheckBox             1
#define gxXdtlEditTextInteger     2
#define gxXdtlEditTextReal        3
#define gxXdtlEditTextString      4
#define gxXdtlPopUp               5
```

**The Panel Information Structure**

```
struct gxPanelInfoRecord {
    gxPanelEvent panelEvt; /* the event */
    short panelResId;      /* resource ID of current panel resource */
    DialogPtr pDlg;        /* pointer to dialog */
    EventRecord *theEvent; /* pointer to event */
    short itemHit;         /* actual item number of event */
    short itemCount;      /* number of items before your items */
    short evtAction;       /* the action that will occur after
                           this event is processed */
    short errorStringId;   /* 'STR ' ID of error string */
}
```

```

gxFormat theFormat;      /* the current format */
void *refCon;            /* refCon from gxPanelSetupRecord */
};

```

## Panel Events

```

enum {
    gxPanelNoEvt      = (gxPanelEvent) 0,      /* no event */
    gxPanelOpenEvt    = (gxPanelEvent) 1,      /* panel is about to open */
    gxPanelCloseEvt   = (gxPanelEvent) 2,      /* panel is about to close */
    gxPanelHitEvt     = (gxPanelEvent) 3,      /* user has selected item */
    gxPanelActivateEvt= (gxPanelEvent) 4,      /* panel has been activated */
    gxPanelDeactivateEvt= (gxPanelEvent) 5,    /* panel has been deactivated */
    gxPanelIconFocusEvt= (gxPanelEvent) 6,     /* focus has changed to icons */
    gxPanelPanelFocusEvt= (gxPanelEvent) 7,    /* focus has changed to panel */
    gxPanelFilterEvt  = (gxPanelEvent) 8,      /* panel event needs to be
                                                filtered */
    gxPanelCancelEvt  = (gxPanelEvent) 9,      /* panel has been canceled */
    gxPanelConfirmEvt = (gxPanelEvent) 10,     /* panel has been confirmed */
    gxPanelDialogEvt  = (gxPanelEvent) 11,     /* panel event to be handled
                                                by the dialog box handler */
    gxPanelOtherEvt   = (gxPanelEvent) 12,     /* an OS event has occurred
                                                in the panel */
    gxPanelUserWillConfirmEvt
                        = (gxPanelEvent) 13    /* user has selected OK */
};

typedef long gxPanelEvent;

```

## Panel Responses

```

enum {
    gxPanelNoResult      = 0,  /* no result from panel */
    gxPanelCancelConfirmation = 1, /* user confirmed panel, but panel
                                    handler discovered an error */
};

typedef long gxPanelResult;

```

## Panel Event Actions

```

enum {
    gxOtherAction      = 0,      /* current item doesn't change after event */
    gxClosePanelAction = 1,      /* panel is closed after event */
};

```



```

    gxCancelDialogAction = 2, /* dialog box is canceled after event */
    gxConfirmDialogAction= 3 /* dialog box is confirmed after event */
};

```

### The Panel Setup Structure

```

struct gxPanelSetupRecord {
    gxPrintingPanelKind    panelKind;          /* kind of program using panel */
    short                  panelResId;         /* resource ID of panel */
    short                  resourceRefNum;     /* resource file refnum of panel */
    void                   *refCon;           /* pointer to panel setup
                                              structure used to build panel */
};

```

### Printing Panel Kinds

```

enum {
    gxApplicationPanel= (gxPrintingPanelKind) 0, /* an application panel */
    gxExtensionPanel   = (gxPrintingPanelKind) 1, /* printing extension panel */
    gxDriverPanel      = (gxPrintingPanelKind) 2 /* printer driver panel */
};

typedef long gxPrintingPanelKind;

```

### Parse Range Results

```

enum {
    gxRangeNotParsed   = (gxParsePageRangeResult) 0, /* not parsed yet */
    gxRangeParsed      = (gxParsePageRangeResult) 1, /* successful parse */
    gxRangeBadFromValue= (gxParsePageRangeResult) 2, /* the "from page" */
                                                         /* value is invalid */
    gxRangeBadToValue  = (gxParsePageRangeResult) 3 /* the "to page" */
                                                         /* value is invalid */
};

typedef long gxParsePageRangeResult;

```

## Functions

---

### Creating and Manipulating Format Objects

```

gxFormat GXNewFormat          (gxJob aJob);
void GXDisposeFormat          (gxFormat aFormat);
gxFormat GXCopyFormat         (gxFormat srcFormat, gxFormat dstFormat);

```

```

gxFormat GXCloneFormat      (gxFormat aFormat);
long GXCountJobFormats      (gxJob aJob);
long GXCountFormatOwners    (gxFormat aFormat);
void GXForEachJobFormatDo    (gxJob aJob, gxFormatProc aFormatProc, void
                             *refCon);

```

### Manipulating Format Object Properties

```

void GXGetFormatMapping      (gxFormat aFormat, gxMapping *aMapping);
gxPaperType GXGetFormatPaperType
                             (gxFormat aFormat);
gxShape GXGetFormatForm      (gxFormat aFormat, gxShape *mask);
void GXSetFormatForm         (gxFormat aFormat, gxShape form, gxShape mask);
void GXChangedFormat         (gxFormat aFormat);

```

### Displaying the Custom Page Setup Dialog Box

```

gxDialogResult GXFormatDialog
                             (gxFormat aFormat,
                              gxEditMenuRecord *anEditMenuRecord,
                              StringPtr title);

```

### Working With Panels

```

void GXSetupDialogPanel      (gxPanelSetupRecord *aPanelSetupRecord);
void GXGetJobPanelDimensions
                             (gxJob aJob, Rect *aRect);
void GXEnableJobScalingPanel
                             (gxJob aJob, Boolean enabled);
short GXGetMessageHandlerResFile
                             (void);

```

### Accessing Printing-Related Collection Objects

```

Collection GXGetJobCollection
                             (gxJob aJob);
Collection GXGetFormatCollection
                             (gxFormat aFormat);
Collection GXGetPaperTypeCollection
                             (gxPaperType aPaperType);

```

## Application-Defined Functions

---

### Message Override Functions for Customizing Dialog Boxes

```

OSErr GXJobPrintDialog      (gxDialogResult *aDialogResult);
OSErr GXJobDefaultFormatDialog
                             (gxDialogResult *aDialogResult);
OSErr GXFormatDialog        (gxFormat aFormat, StringPtr title,
                             gxDialogResult *aDialogResult);
OSErr GXHandlePanelEvent    (gxPanelInfoRecord *aPanelInfoRecord,
                             gxPanelResult *aPanelResult);
OSErr GXFilterPanelEvent    (gxPanelInfoRecord *aPanelInfoRecord,
                             Boolean *returnImmed);
OSErr GXParsePageRange      (StringPtr fromString, StringPtr toString,
                             gxParsePageRangeResult *result);

```

### Looping Through Format Objects

```

gxLoopStatus MyFormatFunction (gxFormat aFormat, void *refCon);

```

## Dialog Box-Related Resources

---

### The Panel Resource

```

type gxPrintPanelType {
    pstring[31];    /* the panel name */
    integer Script; /* script ID */
    fill word;      /* reserve a long word for future use of
                     international */
    fill word;      /* reserve a long word for future use of
                     international */
    integer;        /* the icon ID */
    integer;        /* the item list ID */
};

```

**The Extended Item List Resource**

```

type gxExtendedDITLType {
    integer = $$CountOf(xdtlarray) - 1;
    wide array xdtlarray {
        switch {
            case RadioButtons:
                key      integer = xdtlRadioButtons;
                literal  longint; /* 4 byte id for storage in job
                                object or format object */
                                longint; /* numerical id for storage in
                                job object or format object */
                integer; /* offset in bytes into item */
                integer = $$CountOf(RadioButtonsArray) - 1;
                wide array RadioButtonsArray
                {
                    byte; /* array of corresponding items*/
                };

            case CheckBox:
                key      integer = xdtlCheckBox;
                literal  longint; /* 4-byte ID for storage in job
                                object or format object */
                                longint; /* numerical ID for storage in
                                job object or format object */
                integer; /* offset in bytes into item */
                byte; /* corresponding ditl item */
                fill byte;

            case EditTextInteger:
                key      integer = xdtlEditTextInteger;
                literal  longint; /* 4-byte ID for storage in
                                job object or format object */
                                longint; /* numerical ID for storage in
                                job object or format object */
                integer; /* offset in bytes into item */
                byte; /* corresponding item list's item */
                byte; /* 0 = dont select, 1 = select */
                pstring[15];/* low bound - nil means 'I
                                don't care' */
                pstring[15];/* high bound - nil means 'I
                                don't care' */

```

```

case EditTextReal:
    key      integer = xdtlEditTextReal;
    literal  longint; /* 4-byte ID for storage in job
                       object or format object */
                longint; /* numerical ID for storage in
                       job object or format object */
    integer; /* offset in bytes into item */
    byte; /* corresponding item list's item */
    byte; /* 0 = don't select, 1 = select
pstring[15];/* low bound - nil means 'I
                don't care' */
pstring[15];/* high bound - nil means 'I
                don't care' */

case EditTextString:
    key      integer = xdtlEditTextString;
    literal  longint; /* 4-byte ID for storage in job
                       object */
                       /* or format object */
                longint; /* numerical ID for storage in
                       job object or format object */
    integer; /* offset in bytes into item */
    byte; /* corresponding item list's item */
    byte; /* 0 = don't select, 1 = select */

case PopUp:
    key      integer = xdtlPopUp;
    literal  longint; /* 4-byte ID for storage in job
                       object or format object */
                longint; /* numerical ID for storage in
                       job object or format object */
    integer; /* offset in bytes into item */
    byte; /* corresponding item list's item */
    fill byte;

};
align word;
};
};

```



# Advanced Printing Features

---

## Contents

About Advanced Printing Features	4-5
Printer Objects	4-6
Printer Driver Types	4-7
Printer View Devices	4-8
Color Matching for Printers	4-9
Print File Objects	4-9
Synonyms	4-11
General-Purpose PostScript Operator Synonym	4-12
PostScript Control Information Synonym	4-13
Dash Synonym	4-14
Line Cap Synonym	4-14
Halftone Synonym	4-15
Pattern Synonym	4-17
Cubic Synonym	4-17
QuickDraw Picture Synonym	4-18
Printing Modes	4-19
Pen Tables for Vector Devices	4-20
Using Advanced Printing Features	4-21
Using Advanced Job Object Functions	4-21
Obtaining Printer and Printer Driver Information for a Job	4-22
Getting and Setting the Reference Constant for a Job Object	4-23
Copying Job Object Information	4-25
Working With Printer Objects	4-25
Determining a Printer's Resolution	4-26
Retrieving the Color Profile and Color Space for a Printer	4-27
Manipulating Print File Objects	4-29
Opening and Closing a Print File	4-29
Saving a Print File	4-30
Obtaining the Job Object for a Print File	4-30
Reading Print File Data	4-30

Counting the Pages in a Print File	4-31
Adding or Deleting Print File Pages	4-31
Defining Different Paper Sizes	4-31
Creating a Paper-Type Object	4-32
Obtaining the Name of a Paper Type	4-32
Obtaining the Dimensions of a Paper Type	4-33
Scanning the Paper Types Available to a Job	4-34
Implementing Direct-Mode Printing	4-35
Formatting for Text Job Format Mode Printing	4-36
Using Synonyms	4-38
Advanced Printing Features Reference	4-38
Constants and Data Types for Advanced Printing Features	4-39
Job Format Modes	4-39
Text Job Format (Direct) Mode	4-40
The Status Structure	4-42
Pen Tables for Vector Devices	4-43
Constants and Data Types for Synonyms	4-45
General-Purpose PostScript Operator Synonym	4-45
PostScript Control Information Synonym	4-45
Dash Synonym	4-46
Halftone Synonym	4-46
Line Cap Synonym	4-47
Pattern Synonym	4-47
Cubic Synonym	4-48
QuickDraw Picture Synonym	4-49
Functions	4-49
Advanced Job Object Functions	4-50
GXSelectJobFormattingPrinter	4-50
GXGetJobFormattingPrinter	4-51
GXGetJobOutputPrinter	4-51
GXGetJobRefCon	4-52
GXSetJobRefCon	4-53
GXCopyJob	4-53
Manipulating Printer Objects	4-54
GXGetJobPrinter	4-55
GXGetPrinterJob	4-55
GXForEachPrinterViewDeviceDo	4-56
GXCountPrinterViewDevices	4-57
GXGetPrinterViewDevice	4-57
GXSelectPrinterViewDevice	4-58
GXGetPrinterDriverName	4-59
GXGetPrinterName	4-59
GXGetPrinterDriverType	4-60
GXGetPrinterType	4-61
Working With QuickDraw GX Print Files	4-61
GXOpenPrintFile	4-62
GXClosePrintFile	4-63



GXGetPrintFileJob	4-64
GXCountPrintFilePages	4-65
GXReadPrintFilePage	4-65
GXReplacePrintFilePage	4-66
GXInsertPrintFilePage	4-68
GXDeletePrintFilePageRange	4-69
GXSavePrintFile	4-70
<b>Working With Paper Types</b>	4-71
GXNewPaperType	4-71
GXDisposePaperType	4-72
GXGetNewPaperType	4-73
GXGetJobPaperType	4-74
GXCountJobPaperTypes	4-75
GXCopyPaperType	4-76
GXGetPaperTypeName	4-76
GXGetPaperTypeDimensions	4-77
GXGetPaperTypeJob	4-78
GXForEachJobPaperTypeDo	4-78
<b>Formatting for Specific Devices</b>	4-79
GXSetAvailableJobFormatModes	4-80
GXGetPreferredJobFormatMode	4-80
GXGetJobFormatMode	4-81
GXSetJobFormatMode	4-82
GXJobFormatModeQuery	4-83
<b>Color Profile Functions</b>	4-84
GXFindPrinterProfile	4-84
GXFindFormatProfile	4-85
GXSetPrinterProfile	4-87
GXSetFormatProfile	4-88
<b>Idle Job Function</b>	4-90
GXIdleJob	4-90
<b>Application-Defined Functions</b>	4-90
Message Override Function for the Printing Status Dialog Box	4-90
GXJobStatus	4-91
Looping Through a Printer's View Devices	4-92
Looping Through a Job's Paper Types	4-92
The Status Resource	4-93
Summary of Advanced Printing Features	4-95



This chapter describes how your application can use printing-related objects in ways that may not be required for most applications. Read the information in this chapter if you want your application to read or modify print files after they have been printed, create and use custom paper types, or explicitly control the way that QuickDraw GX performs certain printing operations.

To use this chapter, you should also be familiar with the printing-related objects, including collection objects that QuickDraw GX uses to store job and format information, as introduced in the chapter “Introduction to Printing With QuickDraw GX” in this book. Because the objects and techniques discussed in this chapter build on applications that already provide core printing features, you should be familiar with these features, as introduced in the chapter “Introduction to Printing With QuickDraw GX” and discussed in detail in the “Core Printing Features” chapter of this book.

This chapter describes the concepts required to use advanced QuickDraw GX printing features and terms and then explains how to

- n manipulate a job object; for example, using its reference constant property
- n work with a printer object to obtain information about the device it represents, such as information about the driver, its resolution, and color printing capabilities
- n manipulate a print file object that represents a spooled file or a portable digital document
- n manipulate a paper-type object to define paper sizes for different requirements
- n optimize printing for specific devices

This chapter also describes the resource for Printing Status dialog boxes, as well as status constants. Although you can customize Printing Status dialog boxes in your application, they are used primarily by printer drivers and printing extensions. For information about the use of Printing Status dialog boxes by printer drivers and printing extensions, see the resource chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

## About Advanced Printing Features

---

Advanced printing features make use of objects described in the chapters “Core Printing Features” and “Page Formatting and Dialog Box Customization” in this book. This chapter shows how these objects can be used in additional ways to implement features not typically required by every application that implements QuickDraw GX printing.

For example, the paper-type object is always associated with a format object. A paper type that matches the format is provided by QuickDraw GX as a core feature. Ordinarily, your application need not modify it. If, for example, your application needs to restrict the printable area of a page to reserve room for a letterhead, it can create a paper-type object that defines a new paper size. Although the technique is straightforward, the feature is considered advanced because applications are not required to create

paper-type objects. Typically, the default paper-type object is sufficient for most applications.

To implement other advanced printing features, you use the printer and print file objects.

- n You can use a printer object to determine the device characteristics of a desktop printer, such as its resolution
- n You can use a print file object to determine the contents of a file that has been printed and change them, if you wish.

The following sections describe the printer and print file objects.

## Printer Objects

Each job object references two printer objects. One printer object specifies the output printer on which the document is printed. The other printer object defines the formatting printer that specifies how the document is formatted. A user chooses an output printer in the Print dialog box and a formatting printer in the Page Setup dialog box. When a job object is created, its output printer is the currently selected desktop printer, and the formatting printer is specified by the output printer's printer driver.

Each printer object has six accessible properties, as shown in Figure 4-1. Note that, because a printer object is a private data structure, the order of the properties as shown in Figure 4-1 is completely arbitrary. Properties in italics indicate references to other objects.

**Figure 4-1** The printer object



The properties of a printer object are:

- n **Printer name.** This property contains the name of the printer. A user specifies a printer by name in the Print dialog box. For example, a user could choose the printer “My Printer” from the list of available printers.
- n **Printer type.** This property specifies the creator type of a printer. It is a 4-character signature that uniquely identifies a kind of printer. You are responsible for registering the printer type with Developer Technical Support at Apple Computer. An example of a printer type is 'LWRW' for a LaserWriter.
- n **Printer driver name.** This property specifies the name of the printer driver to which the job is printed. A user specifies a printer driver from the Chooser if the desired printer is not already on the desktop.
- n **Printer driver type.** This property specifies the kind of printer driver. Table 4-1 shows some printer driver types provided by QuickDraw GX.
- n **View device list.** This property contains a list of references to the view devices associated with a printer. Each view device specifies a print resolution (dots-per-inch) and color space (for example, CMYK or a grayscale space) that is supported by the printer. For more information about the relationship between printer objects and view devices, see the section “Printer View Devices” beginning on page 4-8.
- n **Job.** This property contains a reference to a job object. Through this reference, you can access a job object associated with a printer object. The job object is discussed in the chapter “Core Printing Features” in this book.

## Printer Driver Types

Table 4-1 shows the printer driver types defined by QuickDraw GX. Do not make assumptions about the kinds of service provided by a printer driver based on its type alone. For example, two PostScript drivers may be subtly different.

**Table 4-1** Printer driver types

Constant	Value	Explanation
gxAnyPrinterType	'univ'	Universal type of printer
gxRasterPrinterType	'rast'	Raster printer
gxPostscriptPrinterType	'post'	Postscript printer
gxVectorPrinterType	'vect'	Vector printer
gxPortableDocPrinterType	'gxpd'	Portable digital document maker
	'????'	Unknown driver type

### Note

You are responsible for registering your printer driver type with Developer Technical Support. u

## Printer View Devices

---

A printer object's view device list specifies the resolutions and color spaces that can be used with a printer. These view devices are created by the printer driver. Your application can access, but not change, these characteristics. The printer's resolution is stored in the view device's mapping property as the scaling factor. The printer's color space is stored in the bitmap shape that represents the imageable area of the device. A view device object contains other properties as well; however, these properties are not used in printing. For more information about view device objects, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

For example, the LaserWriter IISC GX driver creates a view device list with only one view device, because the printer supports only one color space, black-and-white, and one resolution, 300 dots-per-inch. The view device's mapping property specifies a scaling factor of 4.17, both horizontally and vertically, to achieve the 300 dots-per-inch resolution. The scaling factor is determined by dividing the printer's resolution, 300 dots-per-inch, by 72, which represents the resolution when the scaling factor is 1.

As another example, the ImageWriter II GX printer driver supports printing at two resolutions in each of two color spaces:

- n 144 dpi, with a 4-bit indexed CMYK (cyan, magenta, yellow, black) color space
- n 144 dpi, 1-bit indexed color space
- n 72 dpi, with a 4-bit indexed CMYK color space
- n 72 dpi, 1-bit indexed color space

The driver creates a view device list with four view device references. The printer driver sets up the mapping property in each view device to specify the correct scaling factor. For an example of how to obtain the scaling factor, see the section "Determining a Printer's Resolution" on page 4-26.

### Note

A printer driver inherits a view device for a 72 dpi, 24-bit RGB color space from QuickDraw GX and modifies the list as necessary to include the view devices that the driver actually supports. For more information about writing a printer driver, see the printer driver chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*. u

## Color Matching for Printers

---

QuickDraw GX provides a color profile object that is used to specify color-matching information for a printer. The color profile object is discussed in the color and color-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*. Your application can access the color profile object associated with a printer driver or a particular page of output or set these color profile objects using the following functions:

Function	Purpose
GXFindPrinterProfile	Determine the color profile for a printer
GXFindFormatProfile	Determine the color profile for a format object
GXSetPrinterProfile	Set the color profile for a printer
GXSetFormatProfile	Set the color profile for a format object

For more information about these functions, see “Color Profile Functions” beginning on page 4-84. For an example of retrieving the color profile and color space from a view device, see “Retrieving the Color Profile and Color Space for a Printer” on page 4-27.

## Print File Objects

---

A **print file object** represents the file that QuickDraw GX creates when your application prints a document. If the document is printed to a printer, the print file contains the spooled input to the printer driver. If the document is printed as a portable digital document, the print file’s contents are kept in an application-independent form along with data, such as font information, that allows the document to be viewed without the application that created it.

You can use print file objects to

- n open, save, and close print files
- n retrieve the contents of a print file or the formats associated with it
- n count the pages in a print file, and insert, replace, and delete pages
- n retrieve the job object stored with the print file

Print file objects have four accessible properties, as shown in Figure 4-2. Note, that because a print file object is a private data structure, the order of the properties as shown in Figure 4-2 is completely arbitrary. Properties in italics indicate references to other objects.

**Figure 4-2** The print file object



The properties of a print file object are:

- n **Page count.** This property contains the number of pages in the print file.
- n **Format list.** This property contains a list of references to format objects, one reference for each page in the file. The first reference is the default format for the print job.
- n **Shape list.** This property contains a list of references to shape objects, one reference for each page in the file. Each page is stored as a picture shape in the file, whether the file was created page-by-page or shape-by-shape for each page. Thus, the first reference in the list is the picture shape for the first page, the second reference is the shape for the second page, and so on.
- n **Job.** This property is a reference to the job object associated with the file when it is open. The properties of this job object match those of the job object used to create the print file.



## Synonyms

---

You can use **synonyms** to provide alternative printing directives instead of those generated by QuickDraw GX. You are never required to use a synonym. They are available for you to use if you want to explicitly control the way that QuickDraw GX renders output.

For example, if you have special-purpose PostScript code for printing a shape and wish to use it instead of the PostScript code that QuickDraw GX produces, you can create a synonym for your code and attach it to the shape object. When the shape is printed, the instructions associated with the synonym can be used to render the output.

If you use a synonym, the printer driver also must support its use; otherwise, the synonym is ignored. The synonym is interpreted by the printer driver; thus one printer driver may choose to implement a synonym using PostScript and another printer driver might use a proprietary language to implement the same synonym.

You use a synonym by creating a tag object and setting up a reference to that tag in the shape object or another kind of object. A tag object is a QuickDraw GX object that provides you with the ability to associate data with objects, such as shapes, styles, inks, colors, and transforms. For more information about tag objects, see the tag objects chapter of *Inside Macintosh: QuickDraw GX Objects*.

QuickDraw GX provides five kinds of synonyms:

- n Direct PostScript synonyms, which allow you to explicitly specify PostScript operators for rendering images. You can use these synonyms with shape, style, ink, and transform objects to control the behavior of these objects when printing.
- n Style synonyms, such as dashes, line caps, or patterns that can be associated with style objects.
- n Halftone synonyms, which specify the halftone to be applied when a shape or page is printed. For general information about halftones, see the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*.
- n Cubic synonyms, which provide alternative directives for rendering path shapes. For information about path shapes, see the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.
- n Picture synonyms, which specify QuickDraw picture data for rendering pages. For example, QuickDraw GX uses picture synonyms to spool the output of documents designed for printing with the Macintosh Printing Manager.

### Note

Synonyms remain with the shape or a related object, such as the shape's ink, style, or transform. If you cut or copy a shape and then paste it, the synonyms in the tag objects associated with the shape move with the shape. Synonyms also stay with the shape if the print file that contains the shape is redirected. u

Table 4-2 identifies the synonyms that QuickDraw GX provides.

**Table 4-2** QuickDraw GX printing synonyms

Constant	Value	Explanation
<code>gxPostScriptTag</code>	<code>'post'</code>	Specifies PostScript operators
<code>gxPostControlTag</code>	<code>'psct'</code>	Specifies control information for a PostScript printer
<code>gxDashSynonymTag</code>	<code>'sdsh'</code>	Specifies dashes, for example, with the PostScript <code>setdash</code> operator
<code>gxLineCapSynonymTag</code>	<code>'lcap'</code>	Specifies line caps, for example, with the PostScript <code>setlinecap</code> operator
<code>gxPatternSynonymTag</code>	<code>'ptrn'</code>	Specifies a pattern, for example, on vector devices
<code>gxFormatHalftoneTag</code>	<code>'half'</code>	Specifies halftones, for example, the PostScript halftone graphics state
<code>gxCubicSynonymTag</code>	<code>'cubx'</code>	Specifies a cubic representation for a path
<code>gxQuickDrawPictTag</code>	<code>'pict'</code>	Specifies a shape in QuickDraw picture format

The following sections describe the contents of the tag objects that you create for each of these synonyms. For an example of how to use a synonym, see “Using Synonyms,” which begins on page 4-38.

### General-Purpose PostScript Operator Synonym

If you want QuickDraw GX to use your own PostScript operators for rendering an object, you may create a `gxPostScriptTag` synonym and attach it to the object. If you only need to specify specific operators or set up the halftone graphics state, you may be able to use one of the special-purpose synonyms listed in Table 4-2.

You can reference a tag object that contains the `gxPostScriptTag` synonym from a shape object, style object, ink object, or transform object. The kind of object that references the tag object controls the kind of PostScript operators you can use.

- n With a shape object, you can use PostScript printing operators to render the shape.
- n With a style object, you can use PostScript operators to define all stylistic characteristics for the shapes that refer to the style object.

- n With an ink object, you can use PostScript operators to define the color and transfer mode for the shapes that refer to the ink object.
- n With a transform object, you can use PostScript operators to define the clip and mapping of the shapes that refer to the transform object. The `gxPostScriptTag` synonym may be ignored under certain conditions, such as when a transform object's mapping changes the perspective.

The data in the `gxPostScriptTag` synonym is pure PostScript code that is generated as one continuous PostScript data stream. There is no data type that defines the structure of this synonym. You can attach multiple tag objects to an object. This allows you to distribute data into smaller, more manageable pieces that require less memory to load. For best results, you should limit the data in a `gxPostScriptTag` synonym to 8 KB.

If you choose to write your own PostScript code, it is extremely important to make your PostScript code portable, especially if users create portable digital documents. To create portable PostScript code, try to follow these guidelines:

- n Write PostScript code so that it runs on output devices that support Level 1 and devices that support Level 2.
- n Do not make assumptions about the current “PostScript state” of the output device.
- n Do not make assumptions about the fonts that are installed in the output device.

#### Note

The y-axis of the QuickDraw GX coordinate system is the reverse of the y-axis of the PostScript coordinate system. u

### PostScript Control Information Synonym

---

A shape object can refer to a tag object that contains the `gxPostControlTag` synonym. The synonym includes flags that indicate how to modify the PostScript graphics state.

The `gxPostControlTag` synonym provides data specific to PostScript devices that may be necessary for these devices to properly render the data contained within the `gxPostScriptTag` synonym. You are not required, however, to have a `gxPostControlTag` synonym when you use `gxPostScriptTag` synonyms.

A shape object can refer to, at most, only one `gxPostControlTag` synonym. Information in this synonym affects all `gxPostScriptTag` synonyms attached to a shape object.

The `gxPostControl` structure defines the contents of a `gxPostControlTag` synonym:

```
struct gxPostControl {
    long flags;
};
```

#### Field descriptions

<code>flags</code>	A flag that specifies how a shape is embedded in the PostScript data stream. If it is <code>gxNoSave</code> , the PostScript data should be encapsulated between a save and restore combination. If <code>gxNoSave</code> is not specified or the <code>gxPostControlTag</code> synonym is not present, the save and restore combination is used.
--------------------	---

### Dash Synonym

---

A style object can refer to a tag that contains the `gxDashSynonymTag` synonym. This tag causes QuickDraw GX to print simple dashes. For example, this synonym may cause the printer driver to use the PostScript `setdash` operator instead of the specification in the `dash` property of the style. The phase for the `setdash` operator might still be taken from the phase value stored in the `dash` property of the style object.

The `gxDashSynonym` structure defines the contents of a `gxDashSynonymTag` synonym:

```
struct gxDashSynonym {
    long size;
    fixed dashLength[gxAnyNumber];
};
```

#### Field descriptions

<code>size</code>	The number of elements in a dash array.
<code>dashLength</code>	An array of lengths for the dashes.

### Line Cap Synonym

---

A style object can refer to a tag that contains the `gxLineCapSynonym` synonym. For example, this synonym may cause the printer driver to print with the PostScript `linecap` operator instead of the specification in the `cap` property of the style.

The `gxLineCapSynonym` structure defines the `gxLineCapSynonymTag` synonym:

```
typedef long gxLineCapSynonym;
```

The structure is a long word that specifies one of the values in the `gxLineCaps` enumeration:

```
enum gxLineCaps{
    gxButtCap = 0,
    gxRoundCap = 1,
```

## Advanced Printing Features

```

    gxSquareCap = 2,
    gxTriangleCap = 3
};

```

**Constant descriptions**

<code>gxButtCap</code>	Use a square cap, such as the PostScript butt cap, for the line cap.
<code>gxRoundCap</code>	Use a round cap, such as the PostScript round cap, for the line cap.
<code>gxSquareCap</code>	Use a square cap, such as the PostScript projecting square cap, for the line cap.
<code>gxTriangleCap</code>	Use a triangle cap.

**Halftone Synonym**

---

QuickDraw GX supports halftones to represent more colors than can be represented on a printer by alternating available colors in a fixed cell size to represent more colors. QuickDraw GX, by default, chooses the appropriate halftone for you; however, you can choose to specify halftone information on a shape-by-shape or page-by-page basis yourself.

To provide halftone information for a particular shape object, the shape's ink object must refer to a tag object that contains the `gxFormatHalftoneTag` synonym. This allows halftones to be specified for individual inks. Shapes that are drawn with the same ink use the same halftone. An ink that does not refer to the `gxFormatHalftoneTag` synonym uses the page's halftone.

**Note**

If you specify halftone information on a page-by-page basis, you use the `format-halftone` property in the format collection associated with the page's format. For more information about this property, see the chapter "Page Formatting and Dialog Box Customization" in this book. <sup>u</sup>

The `gxFormatHalftoneInfo` structure defines the contents of a `gxFormatHalftoneTag` synonym:

```

struct gxFormatHalftoneInfo {
    long          numHalftones;
    gxHalftone    halftones[1];
};

```

**Field descriptions**

<code>numHalftones</code>	The number of halftones available for use.
<code>halftones</code>	The array of halftone specifications.

Halftones are specified in the `gxHalftone` structures, which are described completely in the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*:

```
struct gxHalftone{
    fixed          angle;           /* direction of halftone */
    fixed          frequency;       /* cells per inch */
    gxDotType      method;         /* kind of pattern */
    gxTintType     tinting;         /* tint calculation method */
    gxColor        dotColor;       /* color of foreground */
    gxColor        backgroundColor; /* color of background */
    gxColorSpace   tintSpace;      /* color space for tint */
};
```

You can specify any number of halftones. QuickDraw GX selects appropriate halftones from the list of available halftones. Its selection is based upon the `tinting` field in the halftone structures:

- n When you print to a black-and-white PostScript device, QuickDraw GX looks for a halftone structure that specifies `gxLuminanceTint` in the `tinting` field. If no halftone specifies this value, it looks for a halftone specifies `gxComponent4Tint` as its tinting method. Component 4 is the black component in the CMYK (cyan, magenta, yellow, and black) space. If no halftone specifies this tinting method either, the first halftone in the list is used.
- n When you print to a color PostScript device, a maximum of four halftones are used. QuickDraw GX attempts to locate halftones for the following tint calculation methods: `gxComponent1Tint` for the cyan halftone, `gxComponent2Tint` for the magenta halftone, `gxComponent3Tint` field for the yellow halftone, and `gxComponent4Tint` for the black halftone. If a tinting method is in the list more than once, the first one in the list is used.

If a halftone for the `gxComponent4Tint` method is not in the list, QuickDraw GX uses the `gxLuminanceTint` tinting method for the black halftone. If the `gxLuminanceTint` tinting method cannot be found either, QuickDraw GX uses the first halftone in the list for the black halftone.

If QuickDraw GX cannot find a halftone for the `gxComponent1Tint`, `gxComponent2Tint`, or `gxComponent3Tint` tinting methods, it uses the black halftone for the missing tinting method.

It is only possible to use halftones to the extent that a particular PostScript device supports them. The dot color and background color of a halftone are ignored because QuickDraw GX assumes that the dot color for a black-and-white device is black and the dot color for a color device with the `gxComponent2Tint` tinting method is magenta.

#### Note

Continuous tone output devices, such as a 32-bit color printer, may choose to ignore the halftone synonym because halftones are not needed on these output devices. u

## Pattern Synonym

---

A style object can refer to a tag object that contains the `gxPatternSynonymTag` synonym. This synonym causes QuickDraw GX to print with the pattern specified in the tag instead of the specification in the pattern property of the style. For example, vector devices typically support crosshatch patterns.

The `gxPatternSynonymTag` structure defines the contents of a `gxPatternSynonymTag` synonym:

```
struct gxPatternSynonym {
    long    patternType;
    fixed   angle;
    fixed   spacing;
    fixed   thickness;
    gxPoint anchorPoint;
};
```

### Field descriptions

<code>patternType</code>	The pattern type, either <code>gxHatch</code> or <code>gxCrossHatch</code> .
<code>angle</code>	The angle of the lines in the pattern.
<code>spacing</code>	The distance between the lines in the pattern.
<code>thickness</code>	The thickness of the lines in the pattern.
<code>anchorPoint</code>	A point that specifies the upper-left corner at which the pattern begins.

## Cubic Synonym

---

A path shape object can refer to a tag object that contains the `gxCubicSynonymTag` synonym. This synonym causes QuickDraw GX to print with a representation of the shape using cubics, such as Bezier curves, instead of the quadratic Bezier curves specified in the shape's geometry.

The data in this synonym is ignored, however, when

- n it is attached to any shape object other than a path
- n the shape object's transform hierarchy changes the perspective
- n the shape object exceeds the PostScript point limit for the destination device
- n the shape object is used as a pattern, dash, clip, cap, or join

The `gxCubicSynonymTag` synonym contains a stream of flags and points. The flags are specified in the `gxCubicSynonym` enumeration:

```
enum gxCubicSynonym{
    gxIgnoreFlag      = 0x0000,
    gxLineToFlag      = 0x0001,
    gxCurveToFlag     = 0x0002,
    gxMoveToFlag      = 0x0003,
    gxClosePathFlag   = 0x0004
};
```

#### Constant descriptions

<code>gxIgnoreFlag</code>	Ignore this flag; get the next one.
<code>gxLineToFlag</code>	Draw a line from the current point to the point specified after this flag.
<code>gxCurveToFlag</code>	Draw a curve from the current point through the three points specified after this flag.
<code>gxMoveToFlag</code>	Move the start of a new contour, which becomes the current point, to the point specified after this flag.
<code>gxClosePathFlag</code>	Close the contour.

The point, line, or curve specified after a line follow the conventions for a point, line, or curve, (`gxPoint`, `gxLine`, or `gxCurve`), respectively. The rendering of the shape still depends on the fill of the shape object and the shape object's style, ink, and transform.

Each flag is a short integer; however, QuickDraw GX only considers the low 8 bits. Your application can store application-specific flags in the other 8 bits of the word. Set bits that are not used to 0.

### QuickDraw Picture Synonym

---

When QuickDraw GX spools a document containing QuickDraw imaging commands, it creates and flattens, for each page, a QuickDraw GX rectangle shape with an attached tag object of tag type 'pict' (the QuickDraw GX constant for that tag type is `gxQuickDrawPictTag`). The tag object contains information that specifies the characteristics and location of a file containing QuickDraw picture data for that page.

When QuickDraw GX subsequently despools the file, it (or the printer driver) uses the QuickDraw GX Translator to convert the QuickDraw picture data into a QuickDraw GX picture shape before printing it. The tag object contains a `gxQuickDrawPict` structure:

```
struct gxQuickDrawPict {
    gxTranslationOptions    options;
    Rect                   srcRect;
    Point                  styleStretch;
```



## Advanced Printing Features

```

        unsigned long                dataLength;
        struct gxBitmapDataSourceAlias  alias;
};

```

**Field descriptions**

<code>options</code>	The translation options to be used by the QuickDraw GX Translator when converting the QuickDraw data.
<code>srcRect</code>	The source rectangle for the translation, in QuickDraw coordinates. It controls scaling of the image. This rectangle is the QuickDraw picture frame that bounds the QuickDraw data.
<code>styleStretch</code>	The scale factor (both horizontal and vertical) to apply to certain items, such as dashes, in QuickDraw picture comments.
<code>dataLength</code>	The length of the QuickDraw picture data, in bytes.
<code>alias</code>	A structure that defines the location of the file containing the QuickDraw data, and the offset within the file to that data.

The QuickDraw GX rectangle shape that the tag object is attached to specifies the destination bounding rectangle for drawing the QuickDraw data (in QuickDraw coordinates). The relative sizes of the source rectangle and destination rectangle control the scaling of the image when it is translated.

The QuickDraw GX Translator is described in the environment chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. Tag objects are described in the tag objects chapter of *Inside Macintosh: QuickDraw GX Objects*. The `gxBitmapDataSourceAlias` structure is described in the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. QuickDraw picture data is described in *Inside Macintosh: Imaging With QuickDraw*.

## Printing Modes

---

When you print, QuickDraw GX and the printer driver set up your document for printing based on the specifications in the printer driver. For example, if you print to a PostScript printer, QuickDraw GX converts the picture shapes to the appropriate PostScript directives for you—your application does not need to get involved.

There can be cases, however, in which your application may wish to allow the user to specify an alternative way of printing. Thus, the user may choose to print in a **direct mode**, which is a mode that bypasses QuickDraw GX imaging. For example, direct mode may be used in the following cases:

- n to send text to an ImageWriter with built-in fonts
- n to send PostScript-only output; for example, by attaching tag objects to empty shape objects, in which the tag object contains PostScript code

The most common reason that a direct mode may be used is to speed up printing. The major drawback to direct-mode printing is that the user cannot redirect the print file that was created during printing to another printer.

Direct mode is a kind of **job format mode**. QuickDraw GX supports three job format modes, which are shown in Table 4-3. Variables of type `gxJobFormatMode` are used to store the print job format mode.

**Table 4-3** Print job format modes

Constant	Value	Explanation
<code>gxGraphicsJobFormatMode</code>	<code>'grph'</code>	Graphics output, which is used as the default for QuickDraw GX printing
<code>gxTextJobFormatMode</code>	<code>'text'</code>	Text-only output
<code>gxPostScriptJobFormatMode</code>	<code>'post'</code>	PostScript-only output

A printer driver may not support all of these modes, or it may support additional modes that the application and printer driver agree to support. To support a job format mode other than `gxGraphicsJobFormatMode`, the application must specify the available modes. The printer driver uses this list of modes to choose its preferred mode. When the user chooses to use direct mode, the user is selecting the printer driver's preferred mode of printing.

For information about how a printer driver sets the preferred mode, see the printer driver chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*. For an example of how to set the available modes and set the preferred mode in response to the user choosing direct mode, see “Implementing Direct-Mode Printing” on page 4-35.

#### IMPORTANT

Only use `gxTextJobFormatMode` printing when the user requests direct-mode printing. <sup>s</sup>

## Pen Tables for Vector Devices

If a device driver for a vector device sets up a pen table, your application can access it to determine the colors and sizes of the pens in the device's carousel. The driver sets up a pen table in a tag object and creates a reference to the tag object in the view device object associated with the vector device. For more information about how a driver sets up a pen table, see the printing messages chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Your application can reference this pen table by retrieving the contents of the `gxPenTableTag` tag, which is defined as `'pent'`, from the view device object associated with the vector device. For example, if the user creates a line with a thickness that is smaller than a pen's thickness, your application could detect this situation and warn the user that the screen display will not match the printed output.

The `gxPenTableEntry` structure defines the data available for each pen in the carousel:

```
struct gxPenTableEntry {
    Str31    penName;
    gxColor  penColor;
    fixed    penThickness;
    short    penUnits;
    short    penPosition;
};
```

The contents of the `gxPenTableTag` tag object contain one or more of these pen table entries. The contents of the `gxPenTableTag` tag object are defined by a `gxPenTable` structure:

```
struct gxPenTable{
    short          numPens;
    gxPenTableEntry pens[1];
};
```

Several constants are available for comparison with the contents of the `penUnits` field:

- n Use `gxDeviceUnits` to specify device-specific units.
- n Use `gxMmUnits` to specify millimeters.
- n Use `gxInchesUnits` to specify inches.

## Using Advanced Printing Features

---

This section shows you how to implement advanced QuickDraw GX printing features in your application. This section shows you several ways to manipulate job, printer, print file, and paper-type objects. It also shows you how to set up direct-mode printing and use synonyms.

### Using Advanced Job Object Functions

---

QuickDraw GX advanced job object functions allow you to obtain specific information about a particular job object. You can use these functions to

- n retrieve printer driver and device information
- n set or retrieve a job object's reference constant
- n copy job objects

## Obtaining Printer and Printer Driver Information for a Job

---

The job object contains information about the output and formatting printers. You can obtain references to these printer objects with the `GXGetJobOutputPrinter` and `GXGetJobFormattingPrinter` functions, respectively. Listing 4-1 shows how to obtain the reference to the output printer with the `GXGetJobOutputPrinter` function.

You can use these references to call functions to obtain additional information about the printer and its driver from the printer object's properties. Listing 4-1 also shows how to obtain the printer's name using the `GXGetPrinterName` function, the printer driver's name using the `GXGetPrinterDriverName` function, the printer's type using the `GXGetPrinterType` function, and the printer driver's type using the `GXGetPrinterDriverType` function.

For example, you could obtain this information and display it to the user in a dialog box. In this case, you need to convert the printer type and printer driver type to strings. One way you can do this is with the `BlockMove` function, as shown in Listing 4-1.

---

**Listing 4-1**      Obtaining the names and types of a printer and printer driver

```
OSErr MyShowJobPrinterInfo(MyDocumentPtr myDocument)
{
    OSErr      err;
    gxPrinter  jobPrinter;
    OSType     deviceOSType, driverOSType;
    Str255     deviceName, deviceType, driverName, driverType;
    ...
    /*
       Get the current printer for this job. From that, get the
       current device name, driver name, device type, and driver
       type.
    */
    if (err == noErr)
    {
        jobPrinter = GXGetJobOutputPrinter(myDocument->documentJob);

        GXGetPrinterName(jobPrinter, deviceName);
        GXGetPrinterDriverName(jobPrinter, driverName);
        deviceOSType = GXGetPrinterType(jobPrinter);
        driverOSType = GXGetPrinterDriverType(jobPrinter);
    }
}
```

## Advanced Printing Features

```

err = GXGetJobError(myDocument->documentJob);
if (err == noErr)
/*
    Since the device and driver type are OSTypes, convert
    them to the Pascal strings to display.
*/
{
    BlockMove(&deviceOSType, &deviceType[1], (long)
                (deviceType[0] = 4));
    BlockMove(&driverOSType, &driverType[1], (long)
                (driverType[0] = 4));
}
...
return err;
};

```

### Getting and Setting the Reference Constant for a Job Object

---

QuickDraw GX maintains a reference constant in each job object for your application's use. You can use the `GXGetJobRefCon` function to obtain the reference constant and use the `GXSetJobRefCon` function to set it. These functions allow you to associate your own data with a particular job object.

For example, Listing 4-2 shows how you can store a pointer to the document data in the reference constant of a job object for use by an override function that is called by QuickDraw GX.

---

**Listing 4-2**      Setting the job object's reference constant property

```

OSErr MyDoFormatDialog(MyDocumentPtr myDocument)
{
    OSErr          err;
    gxFormat        pageFormat;
    gxDialogResult  result;
    gxEditMenuRecord editMenuRec;
    ...

    /*
        Store the pointer to the document in the job object's
        reference constant to access it within the GXFormatDialog
        override.
    */
    GXSetJobRefCon(myDocument->documentJob, myDocument);
}

```

## Advanced Printing Features

```

    /* Display and handle the Custom Page Setup dialog box. */
    result = GXFormatDialog(pageFormat, nil, &editMenuRec);
    ...
    return err;
}

```

Listing 4-3 shows the override of the `GXFormatDialog` function, in which the format's job object is retrieved. From the job object, the reference constant property is retrieved, allowing access to the document associated with the job object from the override function.

---

**Listing 4-3**      Getting the job object's reference constant property

```

OSError MyFormatDialogOverride(gxFormat aFormat, StringPtr title,
                               gxDialogResult *result)
{
    MyDocumentPtr    myDocument;
    gxJob            formatJob;

    /*
     * Get the current job object by calling GXGetJob. Retrieve the
     * pointer to the document, and use it to set up the dialog box
     * panel.
     */
    formatJob = GXGetJob();
    myDocument = GXGetJobRefCon(formatJob);
    MySetUpPanel(aFormat, myDocument,
                 GXGetMessageHandlerResFile());

    /* Finally, forward this message to other handlers.*/
    return Forward_GXFormatDialog(aFormat, title, result);
}

```

## Copying Job Object Information

---

You can duplicate a job object using the `GXCopyJob` function. This function allows you to take an existing job object that references the output and formatting printers, format object, and other job-specific information, and duplicate it for use with another document. Listing 4-4 shows how to copy the job object from the source document to the destination document. References to formats are no longer valid after you change job objects because the formats are based on another job object. You must set these references to `nil`.

---

**Listing 4-4** Copying job object information

```
OSErr MyCopyJobToDoc(MyDocumentPtr srcDocument, MyDocumentPtr
                    destDocument)
{
    long pg;

    /*
     * Copy the job object information. Note that this changes any
     * formats that the destination job originally had (and the
     * old references become invalid).
     */
    GXCopyJob(srcDocument->documentJob, destDocument->documentJob);

    /* Invalidate any old format object references */
    for (pg = 1; pg <= destDocument->numPages; pg++)
        destDocument->pageFormat[pg - 1] = nil;

    return GXGetJobError(srcDocument->documentJob);
}
```

## Working With Printer Objects

---

Each job object references two printer objects, a formatting printer and an output printer. A printer object is implicitly created by the `GXNewJob` function. There is no external application interface to create or dispose of printer objects.

Examples of how to retrieve a printer object's properties, such as the printer name, printer type, driver name, and driver type are shown in Listing 4-1 on page 4-22. The following sections show you how to obtain the view devices associated with a printer and use them to determine a printer's resolution, color space, and color profile.

## Determining a Printer's Resolution

---

You can determine a printer's resolution from the view devices to which the printer refers. The mapping property of the view device object contains a matrix in which the scaling information is stored. Listing 4-5 shows how to obtain the highest resolution that a printer supports.

---

**Listing 4-5**      Determining a printer's resolution

```
void MyGetFormatDeviceResolution(gxJob whichJob,
                                fixed *hRes, fixed *vRes)
{
    gxPrinter      formatPrinter;
    long           numViewDevices, idx;
    gxViewDevice   printerVDev;
    gxMapping       vDevMapping;

    *hRes = 0;
    *vRes = 0;

    /*
       Get the formatting printer and the number of
       view devices for that printer.
    */
    formatPrinter = GXGetJobFormattingPrinter(whichJob);
    numViewDevices = GXCountPrinterViewDevices (formatPrinter);

    /* Loop through the view devices that this printer supports. */
    for (idx = 1; idx <= numViewDevices; idx++)
    {
        printerVDev = GXGetPrinterViewDevice(formatPrinter, idx);
        GXGetViewDeviceMapping(printerVDev, &vDevMapping);

        if ((vDevMapping.map[0][0] > *hRes) &&
            (vDevMapping.map[1][1] > *vRes))
        {
            *hRes = vDevMapping.map[0][0];
            *vRes = vDevMapping.map[1][1];
        }
    }
}
```



## Advanced Printing Features

```

/*
    Convert scaling factors (multiples of 72 dpi) into
    resolutions.
*/
*hRes = FixedMultiply(*hRes, ff(72));
*vRes = FixedMultiply(*vRes, ff(72));
}

```

### Retrieving the Color Profile and Color Space for a Printer

---

If you wish to retrieve the color profile for a printer, you can call the `GXFindPrinterProfile` function to obtain the reference to a printer's color profile object, or you can call the `GXFindFormatProfile` function to obtain the reference to a format's color profile object. You can set these color profiles with the `GXSetPrinterProfile` and `GXSetFormatProfile` functions, respectively. These functions are described in the reference section of this chapter, starting on page 4-84.

If you want to obtain the color profile of a printer associated with a job object, you can obtain the printer object and, with this reference, you can obtain a reference to the printer's view device. The view device's bitmap shape points to both the color set and the color profile for the printer. Listing 4-6 shows how to retrieve the color profile and color space for the formatting printer.

---

**Listing 4-6**      Retrieving the printer's color profile and color space

```

gxColorProfile MyGetFormattingPrinterProfile
(MyDocumentPtr myDocument, gxColorSpace *theSpace)
{
    gxShape          deviceBitMap;
    gxBitmap         deviceBits;
    gxPrinter        formattingPrinter;
    gxColorProfile    theProfile;
    gxViewDevice      printerDevice;

    /* Get the first profile for the formatting printer. */
    formattingPrinter =
        GXGetJobFormattingPrinter(myDocument->documentJob);
    GXFindPrinterProfile(formattingPrinter, nil, 1, &theProfile);
}

```

## Advanced Printing Features

```

    /*
       Look at the characteristics of the formatting printer's
       view device and retrieve the printer's color space.
    */
    printerDevice = GXGetPrinterViewDevice(formattingPrinter, 0);
    deviceBitMap = GXGetViewDeviceBitmap(printerDevice);
    GXGetBitmap(deviceBitMap, &deviceBits, nil);
    *theSpace = deviceBits.space;
    GXDisposeShape(deviceBitMap);

    return theProfile;
}

```

Listing 4-7 shows how the printer's color profile and color space may be used to determine if a color to be printed is in gamut and to convert the color into the printer's color space.

---

**Listing 4-7** Using the printer's color profile to convert colors

```

Boolean MyMakePrinterColor(gxJob theJob,   gxColor *sourceColor,
                           gxColor *printedColor)
{
    gxColorProfile    printerProfile;
    gxColorSpace      printerSpace;
    Boolean           inGamut;

    /* Get the printer's profile. */
    printerProfile = MyGetFormattingPrinterProfile(theJob,
                                                    &printerSpace);

    /*
       Copy the source color, see if it is in gamut, and convert it
       into the device's color space.
    */
    *printedColor = *sourceColor;
    inGamut = GXCheckColor(printedColor, printerSpace, nil,
                           printerProfile);
    GXConvertColor(printedColor, printerSpace, nil,
                   printerProfile);
    return inGamut;
}

```

**Note**

For more information about colors, color profiles, and color spaces, see the color and color-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*. [u](#)

## Manipulating Print File Objects

---

Print files can only be created by printing, which causes the document to be spooled to the file. A portable digital document is a print file created by the user printing to a PDD Maker GX desktop printer.

After you create a print file, your application or another application can manipulate it. QuickDraw GX allows your application to

- n open and close a print file
- n save a print file
- n retrieve a job object associated with a print file
- n retrieve page or page format data from a print file
- n count the pages in a print file
- n delete, replace, or insert pages

## Opening and Closing a Print File

---

You use the `GXOpenPrintFile` function to open a print file and use the `GXClosePrintFile` function to close one. You must provide a job object when you open the print file. You can dispose of the job object after the file is closed.

Listing 4-8 shows how to open and close a print file. It also shows how to determine the number of pages in a print file with the `GXCountPrintFilePages` function.

---

**Listing 4-8**      Opening and closing a print file

```
OSErr MyGetPrintFilePages(FSSpec *printFSSpec, long *numCopies)
{
    OSErr      err;
    gxPrintFile thePrintFile;
    gxJob       fileJob;

    /*
     * Create a new job object for GXOpenPrintFile, open the print
     * file object, get the number of pages in it, close it, and
     * check for errors. Finally, dispose of the temporary job
     * object and return.
     */
    err = GXNewJob(&fileJob);
```

## Advanced Printing Features

```

    if (err == noErr)
    {
        thePrintFile = GXOpenPrintFile(fileJob, printFSSpec,
                                         fsCurPerm);

        *numCopies = GXCountPrintFilePages(thePrintFile);
        GXClosePrintFile(thePrintFile);

        err = GXGetJobError(fileJob);
        GXDisposeJob(fileJob);
    }

    return err;
}

```

### Saving a Print File

---

You use the `GXSavePrintFile` function to save a print file. You should save a print file after you have added, deleted, or modified its pages, formats, or job object information.

### Obtaining the Job Object for a Print File

---

You use the `GXGetPrintFileJob` function to obtain the job object associated with a particular print file object. This function is useful for determining which job object was associated with the print file when the file was opened by the `GXOpenPrintFile` function, if the reference to the job object was not saved.

### Reading Print File Data

---

You use the `GXReadPrintFilePage` function to retrieve a page from a print file along with its page format. The page is returned as a single picture shape, which is how it is stored in the file, even if the page was created with several calls to `GXDrawShape`.

When you call `GXReadPrintFilePage`, you must specify the page number for the page, starting from 1. You must also specify which view ports you want the picture shape to refer to, so that the shape can be drawn through them when it is displayed onscreen. Listing 4-9 shows how to read a page from a print file.

**Listing 4-9**      Reading a page from a print file

---

```

OSErr MyReadPrintFilePage(MyDocumentPtr myDocument, FSSpec
                           *printFSSpec, long whichPg,
                           gxFormat *pgFormat, gxShape *pgShape)
{
    gxPrintFile    thePrintFile;

    /*
       Open the print file object, read the page, close the file,
       and check for errors.
    */
    thePrintFile = GXOpenPrintFile(myDocument->documentJob,
                                   printFSSpec, fsCurPerm);
    GXReadPrintFilePage(thePrintFile, whichPg, 1,
                        &myDocument->documentViewPort, pgFormat, pgShape);
    GXClosePrintFile(thePrintFile);

    return GXGetJobError(myDocument->documentJob);
}

```

### Counting the Pages in a Print File

---

You use the `GXCountPrintFilePages` function to count the number of pages in the print file object that you specify. See Listing 4-8 on page 4-29 for an example.

### Adding or Deleting Print File Pages

---

After the user prints a file, you can replace, delete, or insert pages. You use the `GXReplacePrintFilePage` function to replace a single page from a print file. You can use the `GXDeletePrintFilePageRange` function to delete a range of pages within a specified print file. You can use the `GXInsertPrintFilePage` function to insert a page in a print file. For changes to the print file to take effect permanently, you must call `GXSavePrintFile` before you call `GXClosePrintFile`.

### Defining Different Paper Sizes

---

QuickDraw GX allows you to define unique paper types for the individual pages of a printable document. You can use the `GXNewPaperType` function to create a new paper-type object for the specified job object, or you can use the `GXGetNewPaperType` function to load a paper-type object from a resource. You use the `GXGetJobPaperType` function to obtain a specific paper-type object by its index into the total set of paper-type object definitions that are accessible from a specific job object. You can use the `GXCountJobPaperTypes` to obtain the total number of paper-type object definitions that are accessible to a particular job object.

## Creating a Paper-Type Object

---

Listing 4-10 shows how to create a new paper-type object. When you create a paper-type object, you specify its name and rectangles that define the paper type's page size and paper size.

---

**Listing 4-10** Creating a new paper-type object

```
OSErr MyCreatePaperType(MyDocumentPtr myDocument, Str31 paperName,
                        gxRectangle *pageSize, gxRectangle *paperSize,
                        gxPaperType *newPaperType)
{
    *newPaperType = GXNewPaperType(myDocument->documentJob,
                                    paperName, pageSize, paperSize);

    return GXGetJobError(myDocument->documentJob);
}
```

You use the `GXDisposePaperType` function to dispose of a paper-type object when it is no longer needed.

## Obtaining the Name of a Paper Type

---

You use the `GXGetPaperTypeName` function to obtain a paper-type object's name. Listing 4-11 shows how to use this function to obtain the name of a paper-type object associated with a format object.

---

**Listing 4-11** Obtaining a paper-type object's name

```
OSErr MyGetPaperTypeName(MyDocumentPtr myDocument, Str255
                        paperTypeName)
{
    gxPaperType thePaperType;
    long        curPage;
    gxFormat    pgFormat;

    /*
     * Get the format object for the current page. If it is nil,
     * you should use the default format.
     */
    curPage = myDocument->curPage;
    pgFormat = myDocument->pageFormat[curPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);
}
```

```

    /* Get the format object's paper-type and name. */
    thePaperType = GXGetFormatPaperType(pgFormat);
    GXGetPaperTypeName(thePaperType, paperTypeName);

    return GXGetJobError(myDocument->documentJob);
}

```

### Obtaining the Dimensions of a Paper Type

---

You use the `GXGetPaperTypeDimensions` function to obtain the page rectangle and the paper rectangle associated with a paper-type object. The page rectangle is the imageable portion of a page. The paper rectangle defines the size of a page. The rectangle size is specified in fixed 72 dpi units. Listing 4-12 shows how to use this function.

---

**Listing 4-12**     Obtaining page and paper rectangles for a paper-type object

```

OSErr MyGetPaperTypeDimensions(MyDocumentPtr myDocument,
                               gxRectangle *pageBounds,
                               gxRectangle *paperBounds)
{
    gxPaperType    thePaperType;
    long           curPage;
    gxFormat       pgFormat;

    /*
     * Get the format object for the current page. If it is nil, use
     * the job object's default format.
     */
    curPage = myDocument->curPage;
    pgFormat = myDocument->pageFormat[curPage - 1];
    if (pgFormat == nil)
        pgFormat = GXGetJobFormat(myDocument->documentJob, 1);

    /*
     * Get the format's paper type and the paper type's bounds.
     * Note that you can also use GXGetFormatDimensions to do this.
     */
    thePaperType = GXGetFormatPaperType(pgFormat);
    GXGetPaperTypeDimensions(thePaperType, pageBounds,
                             paperBounds);

    return GXGetJobError(myDocument->documentJob);
}

```

## Scanning the Paper Types Available to a Job

---

You use the `GXForEachJobPaperTypeDo` function to call an application-defined function for each paper-type object that is accessible to a particular job. The parameters for the `GXForEachJobPaperTypeDo` function, in order, are:

- n the job object whose paper-type objects you wish to examine or change
- n a pointer to the application-defined function you want to execute on these paper-type objects
- n a pointer to a reference constant that refers to additional data you want to make available to the application-defined function
- n a Boolean value that specifies whether you wish to include paper-type objects associated with the formatting printer (`true`) or those associated with the output printer (`false`)

Listing 4-13 shows you how to call an application-defined function, `MyPaperTypeFunction`, for each paper-type object associated with the print job's output printer. The pointer to the reference constant is `nil`.

---

**Listing 4-13** Executing a function for each paper-type object

```
OSErr MyListAllPaperTypes(MyDocumentPtr myDocument)
{
    GXForEachJobPaperTypeDo(myDocument->documentJob,
                           (gxPaperTypeProc) MyPaperTypeFunction, nil,
                           false);
    return GXGetJobError(myDocument->documentJob);
}
```

An application-defined function executed by the `GXForEachJobPaperTypeDo` function is defined as follows:

```
typedef gxLoopStatus (*gxPaperTypeProc) (gxPaperType aPaperType,
                                          void *refCon);
```

The first parameter to the application-defined function is the paper-type object that is to be processed. It is set by the `GXForEachJobPaperTypeDo` function to the next paper-type object automatically. The second parameter is the reference constant passed in by the call to `GXForEachJobPaperTypeDo`. The application-defined function returns a loop status, which it may set to terminate the `GXForEachJobPaperTypeDo` function before every paper-type object has been processed.

Listing 4-14 shows an example of an application-defined function that retrieves the paper type's name and dimensions and can be used to display them. It always returns `gxKeepLooping`, which prevents the `GXForEachJobPaperTypeDo` function from terminating until each paper-type object has been processed.



**Listing 4-14** Executing a procedure for each paper-type object

---

```

pascal gxLoopStatus MyPaperTypeFunction(gxPaperType thePaperType,
                                         void *refCon)
{
    gxRectangle pageBounds, paperBounds;
    Str255      paperTypeName;

    /* Get the paper-type object's name. */
    GXGetPaperTypeName(thePaperType, paperTypeName);

    /* Add code here to display the paper-type object's name. */
    ...

    /* Get the paper-type object's dimensions. */
    GXGetPaperTypeDimensions(thePaperType, &pageBounds,
                             &paperBounds);

    /* Add code here to display the dimensions. */
    ...

    /* Keep looping until all paper types are accessed. */
    return gxKeepLooping;
}

```

## Implementing Direct-Mode Printing

---

Some printer drivers support direct-mode printing, also known as text job format mode printing, in which the generality of QuickDraw GX printing is traded off for faster output using unique features built into the printer hardware. For example, an ImageWriter II printer contains built-in fonts, and its printer driver can make use of them to provide faster printing of text. The printer driver typically allows the user to choose direct-mode printing in these cases.

To allow printing in a nongraphics mode, you must call the `GXSetAvailableJobFormatModes` function to inform the printer driver of all the modes that the application supports, such as `gxGraphicsJobFormatMode` for QuickDraw GX printing, `gxTextJobFormatMode` for direct-mode printing, and `gxPostScriptJobFormatMode` for PostScript-only printing.

All applications should support QuickDraw GX printing. Your application might support direct-mode printing by reformatting the document to match the way it will look when printed, or support PostScript-only output by warning the user that the output cannot be retrieved from a print file when printed in this mode.

**Note**

If you are reformatting the document to match the fonts built into the printer, you must query the printer for the fonts, line lengths, and other information using the `GXJobFormatModeQuery` function. For more information about the information that can be obtained, see the following section, “Formatting for Text Job Format Mode Printing.” <sup>u</sup>

If you want to know the mode in effect after the user dismisses the Page Setup dialog box, you can call `GXGetJobFormatMode`. To change it, you can call `GXSetJobFormatMode`.

## Formatting for Text Job Format Mode Printing

---

If the user chooses to print in a direct mode and the driver’s preferred mode is `gxTextJobFormatMode`, you may choose to reformat the document based on the characteristics of the printer. You must query the printer driver to obtain these characteristics by calling the `GXJobFormatModeQuery` function, which is described on page 4-83.

QuickDraw GX provides an enumerated data type whose values specify the characteristics that you may determine. You use one of these values in the `GXGetJobFormatModeQuery` function to specify the characteristic of interest. Table 4-4 identifies these characteristics. Variables of type `gxQueryType` are used to store the kind of request.

**Table 4-4** Text job format mode query options

---

Constant	Explanation
<code>gxGetJobFormatLineConstraintQuery</code>	Used to determine line constraint characteristics
<code>gxGetJobFormatFontConstraintQuery</code>	Used to determine font positioning constraints
<code>gxGetJobFormatFontCommonStylesQuery</code>	Used to determine the style name, such as “normal” or “bold”
<code>gxSetStyleJobFormatCommonStyleQuery</code>	Used to set style contents
<code>gxGetJobFormatFontsQuery</code>	Used to determine font information

A query returns a pointer to a data structure that contains the requested information. The kind of data structure depends on the kind of query.

The following structures are used to interpret the source and destination data:

- n For the `gxGetJobFormatLineConstraintQuery` query, the source data is nil, and the destination data is returned in a `gxPositionConstraintTable` structure:

```
struct gxPositionConstraintTable {
    gxPoint  phase;
    gxPoint  offset;
    long     numSizes;
    Fixed    sizes[1];
};
```

- n For the `gxGetJobFormatFontConstraintQuery` query, the source data is a `gxFont` reference and the destination data is also returned in a `gxPositionConstraintTable` structure.

#### Note

A `numSizes` value of `gxConstraintRange` indicates a range of sizes, in which `size[0]` specifies the minimum size and `size[1]` specifies the maximum size. u

- n For the `gxGetJobFormatFontCommonStylesQuery` query, the source data is a `gxFont` reference, and the destination data is returned in a `gxStyleNameTable` structure:

```
struct gxStyleNameTable {
    long     numStyleNames;      /* number of style names */
    Str255   styleNames[1];     /* any number of style names */
};
```

- n For the `gxSetStyleJobFormatCommonStyleQuery` query, the source data is a style name from a `gxStyleNameTable` structure, and the destination data is returned in a `gxStyle` reference.
- n For the `gxGetJobFormatFontsQuery` query, the source data is nil, and the destination data is returned in a `gxFontTable` structure:

```
struct gxFontTable {
    long     numFonts;          /* number of font references */
    gxFont   fonts[1];         /* any number of font references */
};
```

## Using Synonyms

---

Synonyms allow you to specify how QuickDraw GX objects are to be printed. Normally, you do not need to use synonyms because QuickDraw GX and the printer driver determine how output is to be rendered and handle it for you. There may be occasions, however, when you want to explicitly specify how an object is to be printed. For example, you might want to specify how to render a path in cubics or explicitly specify the PostScript operators to use when printing an object.

A synonym is stored as a tag object that is referred to by the shapes, inks, transforms, or other objects that use it. There are several ways you can set up your tag object, which are described in the tag objects chapter of *Inside Macintosh: QuickDraw GX Objects*. Whenever you set up your tag, you must specify the tag type and the data itself. For example, the tag type for PostScript is `gxPostScriptTag`. Its data is a stream of PostScript, such as the following:

```
0 0 moveto 10 10 lineto stroke
```

For more information about the synonyms provided by QuickDraw GX, see the section “Synonyms” on page 4-11.

## Advanced Printing Features Reference

---

This section describes the constants, data types, and functions that are specific to advanced printing features of QuickDraw GX.

The Constants and Data Types sections show the enumerations and data types for loop status information for paper-type objects and printer objects, job object direct modes, status dialog box information, paper-type object mapping information, paper-type object view device tag objects, and synonym information.

The “Functions” section describes functions for working with advanced job object functions, manipulating printer objects, working with QuickDraw GX print file objects, working with paper types, and formatting for specific devices.

## Constants and Data Types for Advanced Printing Features

---

This section describes the data types and constants that you use for job format modes, status dialog box information, and pen tables for vector devices.

### Job Format Modes

---

QuickDraw GX provides job format modes that allow a printer driver and an application to negotiate the best mode for printing. The `gxJobFormatMode` data type specifies modes, which are enumerated as follows:

```
enum {
    /* direct modes for job objects */
    gxGraphicsJobFormatMode    = (gxJobFormatMode) 'grph',
    gxTextJobFormatMode        = (gxJobFormatMode) 'text',
    gxPostscriptJobFormatMode  = (gxJobFormatMode) 'post'
};
typedef OSType gxJobFormatMode;
```

#### Constant descriptions

`gxGraphicsJobFormatMode`

If set, QuickDraw GX uses graphics mode.

`gxTextJobFormatMode`

If set, QuickDraw GX uses text mode.

`gxPostScriptJobFormatMode`

If set, QuickDraw GX uses PostScript mode.

The application calls the `GXSetAvailableJobFormatModes` function to inform the printer driver of the modes that the application supports, using a `gxJobFormatModeTable` structure to identify the supported modes.

```
struct gxJobFormatModeTable{
    long            numModes;        /* number of direct modes */
    gxJobFormatMode modes[1];       /* any number direct modes */
};
```

#### Field descriptions

`numModes`

The number of modes that the application supports.

`modes[1]`

An array that contains the modes.

## Text Job Format (Direct) Mode

---

QuickDraw GX provides a text job format mode, sometimes called a direct mode, to format a document to optimize for particular features and capabilities of a device. For example, text mode provides a fast way to print text using the built-in fonts on a device. This feature provides a replacement for draft printing, which was available in previous versions of the printing architecture.

QuickDraw GX defines query types in the query type enumeration to be used with the `gxQueryType` data type:

```
enum {
    /* query types */
    gxGetJobFormatLineConstraintQuery    = (gxQueryType) 0,
    gxGetJobFormatFontsQuery             = (gxQueryType) 1,
    gxGetJobFormatFontCommonStylesQuery  = (gxQueryType) 2,
    gxGetJobFormatFontConstraintQuery    = (gxQueryType) 3,
    gxSetStyleJobFormatCommonStyleQuery  = (gxQueryType) 4
};

typedef long gxQueryType;
```

### Constant descriptions

<code>gxGetJobFormatLineConstraintQuery</code>	Used to determine line constraint characteristics.
<code>gxGetJobFormatFontsQuery</code>	Used to determine font information.
<code>gxGetJobFormatFontCommonStylesQuery</code>	Used to determine the style name, such as “normal” or “bold.”
<code>gxGetJobFormatFontConstraintQuery</code>	Used to determine font positioning constraints.
<code>gxSetStyleJobFormatCommonStyleQuery</code>	Used to set style contents.

QuickDraw GX defines constraint ranges for the constraint table in the constraint range enumeration:

```
enum { gxConstraintRange = -1 };
```

QuickDraw GX stores constraint information in the position constraint table information structure:

```
struct gxPositionConstraintTable {
    gxPoint  phase;
    gxPoint  offset;
    long     numSizes;
    Fixed    sizes[1];
};
```

#### Field descriptions

phase	Where to start from the upper-left corner of the page.
offset	The distance between legal character positions.
numSizes	The number of sizes.
sizes[1]	An array of sizes.

QuickDraw GX stores style information in the style name table information structure:

```
struct gxStyleNameTable{
    long     numStyleNames;
    Str255   styleNames[1];
};
```

#### Field descriptions

numStyleNames	The number of style names.
styleNames[1]	An array of strings containing any number of style names.

QuickDraw GX stores font information in the font table information structure:

```
struct gxFontTable {
    long     numFonts;
    gxFont   fonts[1];
};
```

#### Field descriptions

numFonts	The number of fonts.
fonts	An array of fonts.

## The Status Structure

---

QuickDraw GX defines status type IDs to report various conditions. Not all of these conditions can be reported from the application. For example, although QuickDraw GX defines a status ID for the percentage completion of a print job, it is not available to the application because printing takes place in the background. Status type IDs are specified in the following enumeration:

```
struct gxStatusRecord {
    unsigned short statusType;
    unsigned short statusId;
    unsigned short statusAlertId;
    Signature      statusOwner;
    short          statResId;
    short          statResIndex;
    short          dialogResult;
    unsigned short bufferLen;
    char           statusBuffer[1];
};

typedef struct gxStatusRecord gxStatusRecord;
```

### Field descriptions

<code>statusType</code>	The type of status that this structure represents. This is one of the values shown in Table 4-5.
<code>statusId</code>	The ID of the status that this structure represents. If the value of this field is 0, there is no associated printing alert ('plrt') resource.
<code>statusAlertId</code>	The ID of the printing alert for this status.
<code>statusOwner</code>	The creator type of the owner of this status structure.
<code>statResId</code>	The resource ID for the status ('stat') resource used to process this status.
<code>statResIndex</code>	The index value for indexing into the status resource for this status.
<code>dialogResult</code>	The ID of the button string that was selected to dismiss the printing alert box associated with this status.
<code>bufferLen</code>	The number of bytes in the status buffer.
<code>statusBuffer</code>	This field is a buffer for the caller to store any additional information for use by the status-handling function.

### Note

The triplet of values that includes the `statusOwner`, `statResId`, and `statResIndex` fields must be unique for each status structure. u



Table 4-5 shows the status types that you can specify in a status structure.

**Table 4-5** Status type IDs

Constant	Value	Explanation
<code>gxNonFatalError</code>	1	Affects the icon in the status dialog box
<code>gxFatalError</code>	2	Sends a printing alert to the status dialog box
<code>gxPrinterReady</code>	3	Signals QuickDraw GX to leave alert mode
<code>gxUserAttention</code>	4	Signals initiation of a modal dialog box
<code>gxUserAlert</code>	5	Signals initiation of a printing alert box
<code>gxPageTransmission</code>	6	Signals that a page was sent to the printer and decrements the page counts in strings that are displayed to the user
<code>gxOpenConnectionStatus</code>	7	Signals QuickDraw GX to begin animation on printer icon
<code>gxInformationalStatus</code>	8	Specifies the default status type and has no side effects
<code>gxSpoolingPageStatus</code>	9	Signals that a page was spooled and increments the page count in the status dialog box
<code>gxEndStatus</code>	10	Signals that spooling has ended
<code>gxPercentageStatus</code>	11	Signals QuickDraw GX as to the amount of the job that is currently complete

## Pen Tables for Vector Devices

QuickDraw GX defines a tag object for a paper-type object's view device in the pen table tag enumeration:

```
enum { gxPenTableTag = 'pent' };
```

QuickDraw GX defines paper-type object units in the paper-type units enumeration:

```
enum {
    gxDeviceUnits = 0,
    gxMmUnits     = 1,
    gxInchesUnits = 2
};
```

## Advanced Printing Features

**Constant descriptions**

`gxDeviceUnits`    If set, QuickDraw GX uses specific printer units.

`gxMmUnits`        If set, QuickDraw GX uses millimeters.

`gxInchesUnits`    If set, QuickDraw GX uses inches.

QuickDraw GX defines pen information in the pen not loaded enumeration:

```
enum { gxPenNotLoaded = -1};
```

QuickDraw GX stores pen table information in the pen table information structure:

```
struct gxPenTableEntry {
    Str31      penName;
    gxColor    penColor;
    fixed      penThickness;
    short      penUnits;
    short      penPosition;
};
```

**Field descriptions**

`penName`            A string containing the name of the pen.

`penColor`           The color that is part of the color set.

`penThickness`       The size of the pen.

`penUnits`           The units in which the pen thickness is defined.

`penPosition`        The pen position in the carousel.

QuickDraw GX stores pen information in the pen table information structure:

```
struct gxPenTable {
    short      numPens;
    gxPenTableEntry pens[1];
};
```

**Field descriptions**

`numPens`            The number of pen entries.

`pens[1]`            An array of pen entries.

## Constants and Data Types for Synonyms

---

This section describes the data types and constants that you use for synonyms.

### General-Purpose PostScript Operator Synonym

---

The `gxPostScriptTag` synonym ('post') for the general-purpose PostScript operator is defined:

```
#define gxPostScriptTag    0x706f7374
```

### PostScript Control Information Synonym

---

The `gxPostControlTag` synonym ('psct') for the PostScript control information is defined:

```
#define gxPostControlTag    0x70736374
```

The `gxPostControl` structure defines the contents of a `gxPostControlTag` synonym:

```
struct gxPostControl {
    long flags;
};
```

#### Field descriptions

<code>flags</code>	A flag that specifies how a shape is embedded in the PostScript data stream. If it is <code>gxNoSave</code> , the PostScript data should be encapsulated between a save and restore combination. If <code>gxNoSave</code> is not specified or the <code>gxPostControlTag</code> synonym is not present, the save and restore combination is used.
--------------------	---

QuickDraw GX defines PostScript state flag information in the `gxPsStateFlags` enumeration:

```
enum gxPsStateFlags{
    gxNoSave = 1        /* don't do save-restore around PostScript */
                        /* data */
};
```

## Dash Synonym

---

The `gxDashSynonymTag` **synonym** ('sdsh') for dashes is defined:

```
#define gxDashSynonymTag 0x73647368
```

The `gxDashSynonym` **structure** defines the contents of a `gxDashSynonymTag` **synonym**:

```
struct gxDashSynonym {
    long size;
    fixed dashLength[gxAnyNumber]
};
```

### Field descriptions

<code>size</code>	The number of elements in a dash array.
<code>dashLength</code>	The array of lengths for the dashes.

## Halftone Synonym

---

The `gxFormatHalftoneInfo` **structure** defines the contents of a `gxFormatHalftoneTag` **synonym**:

```
struct gxFormatHalftoneInfo {
    long          numHalftones;
    gxHalftone    halftones[1];
};
```

### Field descriptions

<code>numHalftones</code>	The number of halftones available for use.
<code>halftones</code>	The array of halftone specifications.

**Halftones are specified in the `gxHalftone` structures, which are described completely in the view-related objects chapter of *Inside Macintosh: QuickDraw GX Objects*:**

```
struct gxHalftone{
    fixed          angle;           /* direction of halftone */
    fixed          frequency;       /* cells per inch */
    gxDotType      method;         /* kind of pattern */
    gxTintType     tinting;         /* tint calculation method */
    gxColor        dotColor;        /* color of foreground */
    gxColor        backgroundColor; /* color of background */
    gxColorSpace   tintSpace;       /* color space for tint */
};
```

## Line Cap Synonym

---

The `gxLineCapSynonymTag` synonym ('lcap') for dashes is defined:

```
#define gxLineCapSynonymTag 0x6C636170
```

QuickDraw GX defines line cap information in the line cap synonym enumeration:

```
enum gxLineCaps{
    gxButtCap =      0,
    gxRoundCap =     1,
    gxSquareCap =    2,
    gxTriangleCap =   3
};

typedef long  gxLineCapSynonym;
```

### Constant descriptions

<code>gxButtCap</code>	Use a cap that does not look like a cap, such as the PostScript butt cap.
<code>gxRoundCap</code>	Use a round cap, such as the PostScript round cap.
<code>gxSquareCap</code>	Use a square cap, such as the PostScript projecting square cap.
<code>gxTriangleCap</code>	Use a triangle cap.

## Pattern Synonym

---

The `gxPatternSynonymTag` synonym ('ptrn') for patterns is defined:

```
#define gxPatternSynonymTag 0x7074726E
```

The `gxPatternSynonym` structure defines the contents of a `gxPatternSynonymTag` synonym:

```
struct gxPatternSynonym {
    long    patternType;
    fixed   angle;
    fixed   spacing;
    fixed   thickness;
    gxPoint anchorPoint;
};
```

**Field descriptions**

<code>patternType</code>	The pattern type, either <code>gxHatch</code> or <code>gxCrossHatch</code> .
<code>angle</code>	The angle of the lines in the pattern.
<code>spacing</code>	The distance of the lines in the pattern.
<code>thickness</code>	A point that specifies the upper-left corner at which the pattern begins.

Patterns can be either hatch or crosshatch:

```
enum gxPatterns {
    gxHatch          = 0,
    gxCrossHatch     = 1
};
```

**Constant descriptions**

<code>gxHatch</code>	Use a hatch pattern.
<code>gxCrossHatch</code>	Use a crosshatch pattern.

## Cubic Synonym

---

The `gxCubicSynonymTag` synonym ('cubx') for cubics is defined:

```
#define gxCubicSynonymTag 0x63756278
```

QuickDraw GX defines cubic synonym information in the cubic synonym enumeration:

```
enum gxCubicSynonym{
    gxIgnoreFlag = 0
    gxLineToFlag = 1
    gxCurveToFlag = 2
    gxMoveToFlag = 3
    gxClosePathFlag = 4
};

typedef short gxCubicSynonymFlags;
```

**Constant descriptions**

<code>gxIgnoreFlag</code>	Ignore this flag; get the next one.
<code>gxLineToFlag</code>	Draw a line from the current point to the point specified after this flag.
<code>gxCurveToFlag</code>	Draw a curve from the current point through the three points specified after this flag.
<code>gxMoveToFlag</code>	Move the start of a new contour, which becomes the current point, to the point specified after this flag.
<code>gxClosePathFlag</code>	Close the contour.

## QuickDraw Picture Synonym

---

The `gxQuickDrawPictTag` tag object contains a `gxQuickDrawPict` structure:

```
struct gxQuickDrawPict {
    gxTranslationOptions      options;
    Rect                      srcRect;
    Point                     styleStretch;
    unsigned long             dataLength;
    struct gxBitmapDataSourceAlias alias;
};
```

### Field descriptions

<code>options</code>	The translation options to be used by the QuickDraw GX Translator when converting the QuickDraw data.
<code>srcRect</code>	The source rectangle for the translation, in QuickDraw coordinates. It controls scaling of the image. This rectangle is the QuickDraw picture frame that bounds the QuickDraw data.
<code>styleStretch</code>	The scale factor (both horizontal and vertical) to apply to certain items, such as dashes, in QuickDraw picture comments.
<code>dataLength</code>	The length of the QuickDraw picture data, in bytes.
<code>alias</code>	A structure that defines the location of the file containing the QuickDraw data, and the offset within the file to that data.

## Functions

---

This section describes functions that allow you to implement advanced features of QuickDraw GX printing. Many of these features are implemented by functions that manipulate

- n job objects
- n printer objects and associated view-device objects and color profiles
- n print file objects
- n paper-types objects

Included with each function description is a list of specific result codes returned by QuickDraw GX. In addition to these result codes, you may also receive file-system, memory, and resource errors. For a complete listing of specific file-system, memory, and resource errors, see *Inside Macintosh: C Summary* or *Inside Macintosh: Pascal Summary*.

You should note that not all possible result codes for a particular function are included in function descriptions within this section. For example, the Message Manager, described in *Inside Macintosh: QuickDraw GX Environment and Utilities*, allows QuickDraw GX functions to send specific messages to your application. These messages can also generate errors.

**IMPORTANT**

All printing functions in QuickDraw GX, with the exception of the `GXGetJobError` function, may move Macintosh memory. The `GXGetJobError` function, however, relies on data that may also move. Therefore, your application should never call a QuickDraw GX printing-related function at interrupt time. s

## Advanced Job Object Functions

---

You use the `GXGetJobOutputPrinter` function to determine the output printer for a print job and use the `GXGetJobFormattingPrinter` function to determine the formatting printer for the print job. You use the `GXSelectJobFormattingPrinter` function to specify a formatting printer for a particular print job.

QuickDraw GX provides a place to store a reference constant in each job object for your application's use. A reference constant is accessible through the `GXGetJobRefCon` function. You use the `GXSetJobRefCon` function to set a reference constant.

You can duplicate a job object using the `GXCopyJob` function. This function allows you to take an existing job object and duplicate it for use with another document, causing the associated printer driver, formatting information, and other settings to be used by the other document.

## GXSelectJobFormattingPrinter

---

You can use the `GXSelectJobFormattingPrinter` function to specify a formatting printer for a particular print job.

```
void GXSelectJobFormattingPrinter (gxJob aJob, Str31 printerName);
```

`aJob`                      A reference to the job object for which you are specifying a formatting printer.

`printerName`            The name of the formatting printer.

**DESCRIPTION**

You call `GXSelectJobFormattingPrinter` when the user selects a formatting printer. You can obtain the name of the formatting printer from the Page Setup dialog box and place it in the `printerName` parameter before calling this function.



**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>fnfErr</code>	The printer driver cannot be located.

**GXGetJobFormattingPrinter**

---

You can use the `GXGetJobFormattingPrinter` function to obtain the formatting printer for a particular print job.

```
gxPrinter GXGetJobFormattingPrinter (gxJob aJob);
```

`aJob`            A reference to the job object whose formatting printer you wish to obtain.

*function result* A reference to a printer object.

**DESCRIPTION**

The `GXGetJobFormattingPrinter` function returns a reference to the formatting printer associated with the job specified in the `aJob` parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**GXGetJobOutputPrinter**

---

You can use the `GXGetJobOutputPrinter` function to obtain the output printer for a particular job.

```
gxPrinter GXGetJobOutputPrinter (gxJob aJob);
```

`aJob`            A reference to the job object whose output printer you wish to obtain.

*function result* A reference to a printer object.

**DESCRIPTION**

The `GXGetJobOutputPrinter` function returns a reference to the output printer associated with the job object specified in the `aJob` parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXGetJobOutputPrinter` function, see “Obtaining Printer and Printer Driver Information for a Job” on page 4-22.

**GXGetJobRefCon**

---

You can use the `GXGetJobRefCon` function to obtain a reference constant associated with a particular job object.

```
void* GXGetJobRefcon (gxJob aJob);
```

<code>aJob</code>	A reference to the job object from which you wish to obtain a reference constant.
-------------------	---

**DESCRIPTION**

You can use the `GXGetJobRefCon` function to obtain application-defined data associated with a job object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

To associate a reference constant with a job object, see the description of the `GXSetJobRefCon` function in the next section.

For an example that uses the `GXGetJobRefCon` function, see “Getting and Setting the Reference Constant for a Job Object” on page 4-23.

## GXSetJobRefCon

---

You can use the `GXSetJobRefCon` function to associate a reference constant with a particular job object.

```
void GXSetJobRefcon (gxJob aJob, void *refCon);
```

`aJob`            The job object in which to assign a reference constant.

`refCon`          A pointer to the reference constant to assign.

### DESCRIPTION

The `GXSetJobRefCon` function sets the reference constant for a job object. For example, the reference constant may point to the document data associated with the print job.

### RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

To get the reference constant associated with a job object, see the description of the `GXGetJobRefCon` function in the previous section.

For an example that uses the `GXSetJobRefCon` function, see “Getting and Setting the Reference Constant for a Job Object” on page 4-23.

## GXCopyJob

---

You can use the `GXCopyJob` function to copy job object data from one job object to another.

```
gxJob GXCopyJob (gxJob srcJob, gxJob dstJob);
```

`srcJob`            A reference to the job object to copy.

`dstJob`            A reference to the job object in which to receive the copied data.

*function result*    A reference to a job object.

**DESCRIPTION**

The `GXCopyJob` function makes a copy of the job object specified by the `srcJob` parameter and stores a reference to it in the `dstJob` parameter. If you set the `dstJob` parameter to `nil`, QuickDraw GX allocates and returns a new job object with the properties of the `srcJob` parameter.

For example, you can use this function to copy a job object for use with another document. All information from the source job object is copied into the destination job object, including references to the output and formatting printers, formats, and paper types.

QuickDraw GX allocates appropriate space if the job object that you are copying (the source job object) contains more objects, such as formats, than the job object that you are copying into (the destination job object).

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXCopyJob` function, see “Copying Job Object Information” on page 4-25.

## Manipulating Printer Objects

---

You use the `GXGetJobPrinter` to obtain the printer used by a specific print job. You use the `GXGetPrinterJob` function to obtain the job object associated with a specific printer object.

You use the `GXForEachPrinterViewDeviceDo` function to loop through the view devices associated with a printer object.

You can use the `GXCountPrinterViewDevices` function to obtain the number of view devices associated with a particular printer object.

You use the `GXGetPrinterViewDevice` function to obtain a particular view device associated with a printer object. You use the `GXSelectPrinterViewDevice` function to select the view device to represent a printer’s resolution and color space.

You use the `GXGetPrinterDriverName` and `GXGetPrinterName` functions to obtain the names of a printer and driver, respectively, from a printer object.

You use the `GXGetPrinterDriverType` function to obtain the printer driver type (such as raster, vector, or PostScript) associated with a particular printer object. You use the `GXGetPrinterType` function to obtain the printer’s type.

## GXGetJobPrinter

---

You can use the `GXGetJobPrinter` function to determine the printer object used by a specific job object.

```
gxPrinter GXGetJobPrinter (gxJob aJob);
```

**aJob**            A reference to the job object from which you wish to obtain a printer object.

*function result* A reference to a printer object.

### DESCRIPTION

Your application can use the printer object to determine information specific to a device and printer driver for use in formatting and optimizing the user's data.

### RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

## GXGetPrinterJob

---

You can use the `GXGetPrinterJob` function to obtain the job object associated with a particular printer object.

```
gxJob GXGetPrinterJob (gxPrinter aPrinter);
```

**aPrinter**        A reference to the printer object from which you wish to obtain the job object.

*function result* A reference to the job object associated with the printer object.

### DESCRIPTION

The `GXGetPrinterJob` function returns a reference to the job object that refers to the printer object specified in the `aPrinter` parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**GXForEachPrinterViewDeviceDo**

---

You can use the `GXForEachPrinterViewDeviceDo` function to execute an application-defined function on each view device associated with a particular printer object.

```
void GXForEachPrinterViewDeviceDo (gxPrinter aPrinter,
                                   gxViewDeviceProc aViewDeviceProc,
                                   void *refCon);
```

<code>aPrinter</code>	A reference to the printer object whose view devices you want to manipulate.
<code>aViewDeviceProc</code>	The function you want to execute for each view device.
<code>refCon</code>	A pointer to the reference constant that is passed to the application-defined function.

**DESCRIPTION**

You can use the `GXForEachPrinterViewDeviceDo` function to perform the actions specified in an application-defined function, `aViewDeviceProc`, on all the view devices associated with a particular printer object.

The `GXForEachPrinterViewDeviceDo` function calls your application-defined function and terminates when the application-defined function returns `gxStopLooping` or when `GXForEachPrinterViewDeviceDo` has been called for each view device.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For information about declaring the application-defined function, see “Message Override Function for the Printing Status Dialog Box” on page 4-90.

## GXCountPrinterViewDevices

---

You can use the `GXCountPrinterViewDevices` function to obtain the number of view devices associated with a particular printer object.

```
long GXCountPrinterViewDevices (gxPrinter aPrinter);
```

`aPrinter`     A reference to the printer object whose view devices you want to count.

*function result*     The number of view devices associated with the printer object specified by the `aPrinter` parameter.

### DESCRIPTION

The `GXCountPrinterViewDevices` function returns the number of view devices associated with the specified printer object. A printer object can have multiple view devices, one for each possible combination of printer resolution and color space.

### RESULT CODES

`gxSegmentLoadFailedErr`     A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

For an example that uses the `GXCountPrinterViewDevices` function, see “Determining a Printer’s Resolution” on page 4-26.

## GXGetPrinterViewDevice

---

You can use the `GXGetPrinterViewDevice` function to obtain a printer object’s view device, using an index value.

```
gxViewDevice GXGetPrinterViewDevice (gxPrinter aPrinter,
                                     long whichViewDevice);
```

`aPrinter`     A reference to the printer object whose view device you wish to obtain.

`whichViewDevice`

An index value that specifies the position of the view device reference in the printer object’s view device list.

*function result*     A reference to the specified view device.

**DESCRIPTION**

You specify an index value, starting with 1, in the `whichViewDevice` parameter. The parameter specifies a particular view device. You can specify 0 in the `whichViewDevice` parameter to obtain the view device that represents the current view device, which allows you to obtain the current resolution and color space for the printer.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For examples that use the `GXGetPrinterViewDevice` function, see “Determining a Printer’s Resolution” on page 4-26 and “Retrieving the Color Profile and Color Space for a Printer” on page 4-27.

## **GXSelectPrinterViewDevice**

---

You can use the `GXSelectPrinterViewDevice` function to specify a view device for a printer object.

```
void GXSelectPrinterViewDevice (gxPrinter aPrinter,
                               long whichViewDevice);
```

`aPrinter`     A reference to the printer object associated with a particular view device.

`whichViewDevice`     The index value of the view device you want to select.

**DESCRIPTION**

The `GXSelectPrinterViewDevice` function determines the printer resolution and color space of the printer referenced by the `aPrinter` parameter. A printer object refers to one or more view devices, each of which contains a combination of printer resolution and color space available for the specified printer. You specify an index value, starting with 1, in the `whichViewDevice` parameter. The parameter specifies a particular view device.



**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**GXGetPrinterDriverName**

---

You can use the `GXGetPrinterDriverName` function to obtain the name of the printer driver associated with a particular printer object.

```
void GXGetPrinterDriverName (gxPrinter aPrinter, Str31 name);
```

<code>aPrinter</code>	A reference to the printer object associated with a particular formatting printer driver.
-----------------------	---

<code>name</code>	On return, the formatting printer driver's name.
-------------------	--

**DESCRIPTION**

The `GXGetPrinterDriverName` function retrieves the name of the printer driver to which the `aPrinter` parameter refers and places it in the `name` parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXGetPrinterDriverName` function, see “Obtaining Printer and Printer Driver Information for a Job” on page 4-22.

**GXGetPrinterName**

---

You can use the `GXGetPrinterName` function to obtain the name of the printer associated with a particular printer object.

```
void GXGetPrinterName (gxPrinter aPrinter, Str31 name);
```

<code>aPrinter</code>	A reference to the printer object associated with a printer.
-----------------------	--

<code>name</code>	On return, the printer's name.
-------------------	--------------------------------

**DESCRIPTION**

The `GXGetPrinterName` function retrieves the name of the printer to which the `aPrinter` parameter refers and places it in the `name` parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXGetPrinterName` function, see “Obtaining Printer and Printer Driver Information for a Job” on page 4-22.

## **GXGetPrinterDriverType**

---

You can use the `GXGetPrinterDriverType` function to obtain the printer driver type associated with a particular printer object.

```
OSType GXGetPrinterDriverType (gxPrinter aPrinter);
```

<code>aPrinter</code>	A reference to the printer object associated with a particular printer driver type.
-----------------------	---

*function result* The printer driver type associated with the printer object.

**DESCRIPTION**

The `GXGetPrinterDriverType` function returns a printer type in the format of an `OSType`. Do not make assumptions about the services provided by driver based on its type.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXGetPrinterDriverType` function, see “Obtaining Printer and Printer Driver Information for a Job” on page 4-22.

For possible values of printer driver types, see “Printer Driver Types” on page 4-7.

## GXGetPrinterType

---

You can use the `GXGetPrinterType` function to obtain the printer type of the printer associated with a particular printer object.

```
OSType GXGetPrinterType (gxPrinter aPrinter);
```

`aPrinter`     A reference to the printer object associated with a particular printer.

*function result*   The printer type.

### DESCRIPTION

The `GXGetPrinterType` function returns a printer type in the format of an `OSType`; for example, 'LWRW' for LaserWriter GX.

### RESULT CODES

`gxSegmentLoadFailedErr`     A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

For an example that uses the `GXGetPrinterType` function, see “Obtaining Printer and Printer Driver Information for a Job” on page 4-22.

## Working With QuickDraw GX Print Files

---

You use the `GXOpenPrintFile` and `GXClosePrintFile` functions to open and close print files.

You use the `GXGetPrintFileJob` function to obtain the job object associated with a particular print file. This function is useful when you need to access or modify information of the job object associated with a print file.

You use the `GXCountPrintFilePages` function to count the number of pages in a print file.

You use the `GXReadPrintFilePage` function to retrieve a page or page format for a print file.

You use the `GXReplacePrintFilePage` function to replace a page or page format from a print file. To insert a new page in a print file, you use the `GXInsertPrintFilePage` function.

You use the `GXDeletePrintFilePageRange` function to delete a range of pages within a specified print file.

You use the `GXSavePrintFile` function to save a print file. You should save a print file object if you add, delete, or modify its pages, formats, or job object information.

## **GXOpenPrintFile**

---

You can use the `GXOpenPrintFile` function to open a print file.

```
gxPrintFile GXOpenPrintFile (gxJob aPrintFileJob,
                             FSSpecPtr pFileSpec,
                             char permission);
```

`aPrintFileJob`

A reference to the job object to associate with a particular printer file.

`pFileSpec` A pointer to a file system specification.

`permission`

The access privileges to use when opening the print file object.

*function result* A reference to a print file object.

### **DESCRIPTION**

The `GXOpenPrintFile` function attempts to open the print file specified by a pointer to a file system specification record, `pFileSpec`. If successful, the function returns a print file object that represents the file. The `permission` parameter specifies the access privileges, which can be read-only or read-and-write access.

The information for the print file's job object is unflattened into the job object you specify in the `aPrintFileJob` parameter. This job object specified in the parameter remains associated with the print file until you close the file by calling the `GXClosePrintFile` function.

To check for errors, you should call the `GXGetJobError` function with the specified job object following calls that operate on the print file.

### **SPECIAL CONSIDERATIONS**

The `GXOpenPrintFile` function sets up a warning handler, which chains to the application's warning handler, if it exists. For more information about warning handlers, see the errors, warnings, and notices chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxIncompletePrintFileErr</code>	Contents of file are incomplete.
<code>gxCrashedPrintFileErr</code>	File is currently printing or crashed while printing.
<code>gxInvalidPrintFileVersion</code>	Cannot read file due to incompatible file version.
<code>gxFlattenVersionTooNew</code>	An attempt was made to unflatten a job object that was flattened using a later version of QuickDraw GX.
<code>collectionVersionErr</code>	The version of the collection object is not compatible with the current version of the Collection Manager.

**SEE ALSO**

For an example that uses the `GXOpenPrintFile` function, see “Opening and Closing a Print File” on page 4-29.

To close a print file object, you use the `GXClosePrintFile` function, which is described in the next section.

**GXClosePrintFile**

---

You can use the `GXClosePrintFile` function to close a print file and invalidate the reference to the print file object.

```
void GXClosePrintFile (gxPrintFile aPrintFile);
```

`aPrintFile`

A reference to the print file object for the file to close.

**DESCRIPTION**

The `GXClosePrintFile` function closes the specified file and invalidates the print file object’s association with a job object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXClosePrintFile` function, see “Opening and Closing a Print File” on page 4-29.

**GXGetPrintFileJob**

---

You can use the `GXGetPrintFileJob` function to obtain the job object associated with a particular print file object.

```
gxJob GXGetPrintFileJob (gxPrintFile aPrintFile);
```

`aPrintFile`

A reference to the print file object whose job object you wish to obtain.

*function result* A reference to a job object.

**DESCRIPTION**

The `GXGetPrintFileJob` function returns a reference to the job object that was associated with the print file object when you called the `GXOpenPrintFile` function. If you save the reference when you call the `GXOpenPrintFile` function, you do not need to call this function.

This function is useful when you need to access or modify information in the job object associated with a print file object. For example, you can use this function to obtain the job object and then call `GXGetJobError` for the job object to test for an error condition associated with the print file.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## GXCountPrintFilePages

---

You can use the `GXCountPrintFilePages` function to count the number of pages in a print file.

```
long GXCountPrintFilePages (gxPrintFile aPrintFile);
```

`aPrintFile`

A reference to the print file object that represents the print file.

*function result* The number of pages in the file.

### DESCRIPTION

The `GXCountPrintFilePages` function returns the number of pages in the file.

### RESULT CODES

`gxSegmentLoadFailedErr`

A required code segment could not be found, or there was not enough memory to load it.

## GXReadPrintFilePage

---

You can use the `GXReadPrintFilePage` function to retrieve a page or page format for a print file object.

```
void GXReadPrintFilePage (gxPrintFile aPrintFile, long pageNumber,
                          long numViewPorts, gxViewPort *viewPortList,
                          gxFormat *pageFormat, gxShape *pageShape);
```

`aPrintFile`

A reference to the print file object whose file you want to access.

`pageNumber`

The page you want to access.

`numViewPorts`

The number of view ports in the view port list.

`viewPortList`

A pointer to a list of references to view ports through which you want the page's picture shape to draw.

`pageFormat`

On return, a reference to the format object associated with the page.

`pageShape`

On return, a reference to the picture shape that contains the page's data.

**DESCRIPTION**

The `GXReadPrintFilePage` function retrieves the print file object's page that you specify in the `pageNumber` parameter. It returns the page format and a picture shape representing the contents of the page in the `pageFormat` and `pageShape` parameters, respectively. You can set one or both of these parameters to `nil` if you do not want them returned.

The page shape is associated with the view ports in the `viewPortList` list parameter, which is the list of view ports you want the shape to be drawn through when you call `GXDrawShape` for the shape in the `pageShape` parameter. The `numViewPorts` parameter specifies how many view ports are in the list.

**SPECIAL CONSIDERATIONS**

Do not change the page format or page shape, pointed to by the `pageFormat` and `pageShape` parameters, directly. If you want to change the format or shape, make a copy of the format or shape and modify the copy. After you make a change to the copy, you can replace the format or page in the print file with your copy or insert your copy into the print file.

For speed and memory efficiency, dispose of the references to the format and page shape objects as soon as they are no longer needed. For example, dispose of them as soon as you make a copy of them or draw a page with them.

The page number specified in the `pageNumber` parameter must be valid. Call the `GXCountPrintFilePages` function to ensure that the page number is valid.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For an example that uses the `GXReadPrintFilePage` function, see “Reading Print File Data” on page 4-30.

## **GXReplacePrintFilePage**

---

You can use the `GXReplacePrintFilePage` function to replace a page in a print file object.

```
void GXReplacePrintFilePage (gxPrintFile aPrintFile,
                             long pageNumber, gxFormat pageFormat,
                             gxShape pageShape);
```



## Advanced Printing Features

<code>aPrintFile</code>	A reference to the print file object in which you want to replace a page.
<code>pageNumber</code>	The page you want to replace.
<code>pageFormat</code>	A reference to the page's format object.
<code>pageShape</code>	A reference to the page's picture shape object.

**DESCRIPTION**

The `GXReplacePrintFilePage` function replaces in the page specified in the `pageNumber` parameter.

You specify a replacement page format and page shape in the `pageFormat` and `pageShape` parameters, respectively. You can specify `nil` for either of these parameters to ensure that the page format or the page shape remains unchanged.

Any changes you make to a print file are not permanent until you save the print file object with the `GXSavePrintFile` function.

**SPECIAL CONSIDERATIONS**

After you call the `GXReplacePrintFilePage` function, do not change the page format or page shape referenced by the `pageFormat` and `pageShape` parameters. For example, if you want to change the format or shape later, make a copy, and modify the copy. Dispose of the original page or format after you make the copy.

For speed and memory efficiency, dispose of the references to the format and page parameters immediately after you call `GXReplacePrintFilePage`.

If a format or page is to be duplicated, passing a clone of the object to the function is more efficient than passing a copy. For example, you can pass a clone of a page or format to replicate a page or format already in the file. The cloned object may be one that you have previously read from a print file or one that you created.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

To save a print file object, see the description of the `GXSavePrintFile` function on page 4-70.

## GXInsertPrintFilePage

---

You can use the `GXInsertPrintFilePage` function to insert a new page in a print file.

```
void GXInsertPrintFilePage (gxPrintFile aPrintFile,
                           long atPageNumber, gxFormat pageFormat, gxShape pageShape);
```

`aPrintFile`

A reference to the print file object in whose file you want to insert a page.

`atPageNumber`

The page to insert.

`pageFormat`

A reference to a format object for the inserted page.

`pageShape`

A reference to a picture shape object for the inserted page.

### DESCRIPTION

The `GXInsertPrintFilePage` function inserts a page in a print file before the page number that you specify in the `atPageNumber` parameter. You can pass a value of 1 in this parameter to insert the new page before all other pages in the print file. When you pass a value that is higher than the current page count, QuickDraw GX appends the page to the end of the print file.

Any changes you make to a print file are not permanent until you save the print file object by calling the `GXSavePrintFile` function.

### SPECIAL CONSIDERATIONS

After you call the `GXInsertPrintFilePage` function, do not change the page format or page shape referenced by the `pageFormat` and `pageShape` parameters. For example, if you want to change the format or shape later, make a copy, and modify the copy. Dispose of the original page or format after you make the copy.

For speed and memory efficiency, dispose of the references to the format and page parameters immediately after you call `GXInsertPrintFilePage`.

If a format or page can be reused, passing a clone of the object to the function is more efficient than passing a copy. For example, you can pass a clone of a page or format to replicate a page or format already in the file. The cloned object may be one that you have previously read from a print file or one that you created.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

To save a print file object, see the description of the `GXSavePrintFile` function on page 4-70.

## **GXDeletePrintFilePageRange**

---

You can use the `GXDeletePrintFilePageRange` function to delete a range of pages within a particular print file object.

```
void GXDeletePrintFilePageRange (gxPrintFile aPrintFile,
                                long fromPageNumber,
                                long toPageNumber);
```

`aPrintFile`

A reference to the print file object from whose file you want to delete pages.

`fromPageNumber`

The first page that you want to delete.

`toPageNumber`

The last page that you want to delete.

**DESCRIPTION**

The `GXDeletePrintFilePageRange` function deletes a page or pages in a print file object within the range that you specify in the `fromPageNumber` and `toPageNumber` parameters. The range of page numbers is inclusive. For example, deleting from page 2 to page 3 deletes both pages 2 and 3.

Any changes you make to a print file are not permanent until you save the print file object with the `GXSavePrintFile` function.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

To save a print file object, see the description of the `GXSavePrintFile` function in the next section.

**GXSavePrintFile**

---

You can use the `GXSavePrintFile` function to save a print file object.

```
void GXSavePrintFile (gxPrintFile aPrintFile, FSSpec *pFileSpec);
```

`aPrintFile`

A reference to the print file object whose file you want to save.

`pFileSpec`

A pointer to a file system specification record.

**DESCRIPTION**

The `GXSavePrintFile` function writes an entire print file to disk. This file must previously have been opened with the `GXOpenPrintFile` function. To replace or update the print file, you can pass `nil` in the `pFileSpec` parameter. Otherwise, you can specify a name and location in the `pFileSpec` parameter to save the updated print file and leave the original print file intact.

This function compacts a print file by recovering any space no longer needed. Space becomes available when pages are removed or when a format no longer references any pages. This function also permanently saves any changes that you have made to the print file.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## Working With Paper Types

---

You use the `GXNewPaperType` function to create a new paper-type object, and you use the `GXDisposePaperType` function to dispose of a paper-type object.

You can use the `GXGetNewPaperType` function to retrieve a paper-type object from a resource or use the `GXGetJobPaperType` function to access a specific paper-type object. You use the `GXGetJobPaperType` function to obtain the indexed paper-type object from the total set of paper-type objects that are accessible to a particular job object.

You can use the `GXCountJobPaperTypes` function to obtain the total number of paper-type definitions that are accessible to a particular job object.

You use the `GXCopyPaperType` function to replace the contents of the destination paper-type object with that of the source paper-type object.

You use the `GXGetPaperTypeName` function to obtain the name of a paper-type object.

You use the `GXGetPaperTypeDimensions` function to obtain the page rectangle and the paper rectangle associated with a paper-type object.

You use the `GXGetPaperTypeJob` function to obtain the reference to the job object that owns the paper-type object.

You use the `GXForEachJobPaperTypeDo` function to call an application-defined function for each paper-type definition that is accessible to a particular job object.

## GXNewPaperType

---

You can use the `GXNewPaperType` function to create a new paper-type object.

```
gxPaperType GXNewPaperType (gxJob aJob, Str31 name,
                             gxRectangle *pageSize, gxRectangle *paperSize);
```

<code>aJob</code>	A reference to the job object with which to associate the new paper-type object.
<code>name</code>	The name of the new paper type.
<code>pageSize</code>	A pointer to a rectangle that defines the page size, or imageable area of the paper.
<code>paperSize</code>	A pointer to a rectangle that defines the paper size.

*function result* A reference to the newly created paper-type object.

### DESCRIPTION

The `GXNewPaperType` function creates a paper-type object with the title `name`, the imageable area defined by the `pageSize` rectangle, and the paper size defined by the `paperSize` rectangle. This function associates a paper type of these specifications with the specified job object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>collectionVersionErr</code>	The version of the collection object is not compatible with the current version of the Collection Manager.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

**SEE ALSO**

For an example that uses the `GXNewPaperType` function, see “Creating a Paper-Type Object” on page 4-32.

## **GXDisposePaperType**

---

You can use the `GXDisposePaperType` function to dispose of a paper-type object.

```
void GXDisposePaperType (gxPaperType aPaperType);
```

`aPaperType`

A reference to the paper-type object that you want to dispose of.

**DESCRIPTION**

The `GXDisposePaperType` function disposes of the paper-type object specified by the `aPaperType` parameter by decrementing its owner count. If the owner count falls to 0, QuickDraw GX may delete the paper-type object.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## GXGetNewPaperType

---

You can use the `GXGetNewPaperType` function to create a new paper-type object from a resource template.

```
gxPaperType GXGetNewPaperType (gxJob aJob, short resID);
```

`aJob`            A reference to the job object associated with the new paper-type object.

`resID`           The ID of the resource template.

*function result* A reference to a paper-type object.

### DESCRIPTION

The `GXGetNewPaperType` function creates a paper-type object in the same way that the `GXNewPaperType` function does, except that the title, the imageable area, and the paper size are defined in the resource identified by `resID`. The `GXGetNewPaperType` function associates the returned paper-type object reference with the `aJob` job object.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>collectionVersionErr</code>	The version of the collection object is not compatible with the current version of the Collection Manager.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

## GXGetJobPaperType

---

You can use the `GXGetJobPaperType` function to access the specified paper-type object associated with a particular job object.

```
gxPaperType GXGetJobPaperType (gxJob aJob, long whichPaperType,
                                Boolean forFormatDevice,
                                gxPaperType aPaperType);
```

`aJob`                   A reference to the job object from which to obtain the paper-type object.

`whichPaperType`       The index that specifies which paper-type object to obtain.

`forFormatDevice`      A Boolean value that specifies whether the paper-type objects are

associated with the formatting printer (`true`) or with the output printer (`false`).

`aPaperType`           A valid paper-type object reference.

*function result*   A reference to a paper-type object.

### DESCRIPTION

The `GXGetJobPaperType` function retrieves the specified paper type from the job object based on the index value in the `whichPaperType` parameter. Index values begin at 1.

Set the `forFormatDevice` parameter to `true` to retrieve only the paper types associated with the formatting printer or to `false` to retrieve only paper types associated with the output printer.

If the desired paper-type object is found, based on its index value, this function replaces the contents of the `aPaperType` parameter with that of the retrieved paper-type object.

If the paper-type object is not located, the job object's error is set to `gxPaperTypeNotFound`. Any error generated by this function can be retrieved using the `GXGetJobError` function.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.



## GXCountJobPaperTypes

---

You can use the `GXCountJobPaperTypes` function to obtain the total number of paper-type definitions that are accessible to a particular job object.

```
long GXCountJobPaperTypes (gxJob aJob, Boolean forFormatDevice);
```

**aJob**            A reference to the job object from which to obtain the number of paper-type definitions.

**forFormatDevice**    A Boolean value that specifies whether the paper-type objects are associated with the formatting printer (`true`) or with the output printer (`false`).

*function result*    The number of paper-type objects that are associated with the print job.

### DESCRIPTION

The `GXCountJobPaperTypes` function returns the number of paper types associated with either the print job's formatting printer or output printer.

Set the `forFormatDevice` parameter to `true` to count only the paper types associated with the formatting printer or to `false` to count only paper types associated with the output printer.

Depending on the format specification of the job object, the total number of paper types returned may include the total number of system paper types, user paper types, printer driver paper types, and printer-configuration-file paper types.

Use the `GXGetJobError` function to retrieve errors for this function.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## GXCopyPaperType

---

You can use the `GXCopyPaperType` function to copy paper-type object data from one paper-type object to another paper-type object.

```
gxPaperType GXCopyPaperType (gxPaperType srcPaperType,
                             gxPaperType dstPaperType);
```

`srcPaperType`

A reference to the paper-type object whose data you want to copy.

`dstPaperType`

A reference to the paper-type object in which to copy the data.

*function result* Reference to a paper-type object.

### DESCRIPTION

The `GXCopyPaperType` function copies the contents of the paper-type object referred to in the `srcPaperType` parameter to the paper-type object referred to in the `dstPaperType` parameter. Each component of the paper-type object is copied. You must specify valid paper types in both the `srcPaperType` and `dstPaperType` parameters.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

## GXGetPaperTypeName

---

You can use the `GXGetPaperTypeName` function to obtain the name of a paper-type object.

```
void GXGetPaperTypeName (gxPaperType aPaperType,
                        Str31 name);
```

`aPaperType`

A reference to the paper-type object from which to obtain the name.

`name`

On return, the name of the paper type.

**DESCRIPTION**

The `GXGetPaperTypeName` function returns the name of the paper-type object specified by the `aPaperType` parameter. The `aPaperType` parameter must refer to a valid paper-type object. The name of the paper-type object is returned in the `name` parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

**SEE ALSO**

For an example that uses the `GXGetPaperTypeName` function, see “Obtaining the Name of a Paper Type” on page 4-32.

## **GXGetPaperTypeDimensions**

---

You can use the `GXGetPaperTypeDimensions` function to obtain the page rectangle and the paper rectangle associated with a paper-type object.

```
void GXGetPaperTypeDimensions (gxPaperType aPaperType,
                               gxRectangle *aPageSize,
                               gxRectangle *aPaperSize);
```

`aPaperType`

A reference to the paper-type object from which to obtain page and paper sizes.

`aPageSize`

A pointer to a rectangle that receives the page size of the paper type.

`aPaperSize`

A pointer to a rectangle that receives the paper size of the paper type.

**DESCRIPTION**

The `GXGetPaperTypeDimensions` function returns the page and paper size for the specified paper type in the geometry of rectangles. The page rectangle is the imageable portion of a page. The paper rectangle is the size of the paper. The geometry for each rectangle specifies the size in 72 dots-per-inch units. Passing a `nil` pointer for either the `aPageSize` or the `aPaperSize` parameters causes QuickDraw GX to ignore the parameter.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPaperTypeNotFound</code>	The paper-type object cannot be located.

**SEE ALSO**

For an example that uses the `GXGetPaperTypeDimensions` function, see “Obtaining the Dimensions of a Paper Type” on page 4-33.

## **GXGetPaperTypeJob**

---

You can use the `GXGetPaperTypeJob` function to obtain a reference to the job object that owns a paper-type object.

```
gxJob GXGetPaperTypeJob (gxPaperType aPaperType);
```

`aPaperType`

A reference to the paper-type object for which you want to obtain the job object.

*function result* A reference to the job object that owns the paper type.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## **GXForEachJobPaperTypeDo**

---

You can use the `GXForEachJobPaperTypeDo` function to call an application-defined function for each paper-type definition that is accessible to a particular job object.

```
void GXForEachJobPaperTypeDo (gxJob aJob,
                             gxPaperTypeProc aPaperTypeProc,
                             void *refCon,
                             Boolean forFormattingPrinter);
```

`aJob`

A reference to the job object from which to obtain the paper-type object.

`aPaperTypeProc`

An application-defined function to be called for each paper-type definition accessible to a job object.

## Advanced Printing Features

`refCon`           A pointer to a reference constant.

`forFormattingPrinter`           A Boolean value that specifies whether the paper-type objects are associated with the formatting printer (`true`) or with the output printer (`false`).

### DESCRIPTION

The `GXForEachJobPaperTypeDo` function loops over each of the paper-type objects for the specified print job, executing the application-supplied function on each one.

The application-defined function is called until either all the paper types have been processed or the function returns the `gxStopLooping` constant.

Set the `forFormattingPrinter` parameter to `true` to execute the application-defined function only on the paper types associated with the formatting printer or to `false` to execute the application-defined function only on paper types associated with the output printer.

### RESULT CODES

`gxSegmentLoadFailedErr`           A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

For an example that uses the `GXForEachJobPaperTypeDo` function, see “Scanning the Paper Types Available to a Job” on page 4-34.

For information about declaring the application-defined function, see “Looping Through a Job’s Paper Types” on page 4-92.

## Formatting for Specific Devices

---

You use the `GXSetAvailableJobFormatModes` function to set your list of job format modes for a particular job object, and you use the `GXGetPreferredJobFormatMode` function to obtain the printer driver’s preferred mode.

You use the `GXGetJobFormatMode` function to obtain the current job format mode and the `GXSetJobFormatMode` function to set it.

You use the `GXJobFormatModeQuery` function to get or set additional information for the text job format mode.

## GXSetAvailableJobFormatModes

---

You can use the `GXSetAvailableJobFormatModes` function to set the list of job format modes that your application supports.

```
void GXSetAvailableJobFormatModes (gxJob aJob,
                                   gxJobFormatModeTableHdl aJobFormatModeTableHdl);
```

`aJob`                   A reference to the job object to which the list of format modes applies.

`aJobFormatModeTableHdl`

A handle that contains the list of supported modes.

### DESCRIPTION

The `GXSetAvailableJobFormatModes` function provides the printer driver with the list of modes that the printer driver could return as its preferred mode.

### RESULT CODES

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

### SEE ALSO

For more information about how to use this function, see “Implementing Direct-Mode Printing” on page 4-35.

## GXGetPreferredJobFormatMode

---

You can use the `GXGetPreferredJobFormatMode` function to obtain the preferred mode of printing to the printer associated with a print job.

```
gxJobFormatMode GXGetPreferredJobFormatMode (gxJob aJob,
                                              Boolean *directOnly);
```

`aJob`                   A reference to the job whose format mode you wish to determine.

`directOnly`

A pointer to a Boolean value returned by this function that specifies whether the preferred mode is the only mode.

*function result*   The preferred mode.

**DESCRIPTION**

The `GXGetPreferredJobFormatMode` function returns the preferred mode of printing to the job's output printer. The preferred mode is one of the modes proposed by the application in a call to `GXSetAvailableJobFormatModes`. From that information, the printer driver can respond with its preferred mode.

The preferred mode is typically a mode supported directly by the driver's hardware. In the case of an ImageWriter II, the `GXGetPreferredJobFormatMode` function returns `gxTextJobFormatMode` because it can use fonts built into the printer itself for faster text printing. The preferred mode typically represents the job format mode with the fastest throughput; however, it may limit the quality or even the kind of output that may be printed.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

**SEE ALSO**

For more information about how to use this function, see "Implementing Direct-Mode Printing" on page 4-35.

## **GXGetJobFormatMode**

---

You can use the `GXGetJobFormatMode` function to obtain the current job format mode for a particular job object.

```
gxJobFormatMode GXGetJobFormatMode (gxJob aJob);
```

<code>aJob</code>	A reference to the job object whose current format mode you wish to obtain.
-------------------	---

*function result* The current job format mode.

**DESCRIPTION**

The `GXGetJobFormatMode` function returns the current job format mode specified in the `GXSetJobFormatMode` function. The modes defined by QuickDraw GX are:

Constant	Value	Explanation
<code>gxGraphicsJobFormatMode</code>	<code>'grph'</code>	QuickDraw GX default printing
<code>gxTextJobFormatMode</code>	<code>'text'</code>	Text-only output
<code>gxPostScriptJobFormatMode</code>	<code>'post'</code>	PostScript-only output

A printer driver may define additional modes.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## **GXSetJobFormatMode**

---

You can use the `GXSetJobFormatMode` to set the job format mode.

```
void GXSetJobFormatMode (gxJob aJob, gxJobFormatMode aMode);
```

<code>aJob</code>	A reference to the job object associated with the direct mode.
<code>aMode</code>	The direct mode to set.

**DESCRIPTION**

The `GXSetJobFormatMode` function activates the specified job format mode for a job object whether or not the mode is supported by the printer driver or the application. You might want to call `GXSetJobFormatMode` to set the mode when printing without dialog boxes, such as when the user prints from the Finder.

**RESULT CODES**

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--



## GXJobFormatModeQuery

---

You can use the `GXJobFormatModeQuery` function to get or set additional information related to a job format mode.

```
void GXJobFormatModeQuery (gxJob aJob, gxQueryType aQueryType,
                           void *srcData, void *dstData);
```

<code>aJob</code>	A reference to the job object for which information relating to the printer's format mode is being requested.
<code>aQueryType</code>	The kind of query requested.
<code>srcData</code>	A pointer to the source data.
<code>dstData</code>	A pointer to the destination data.

### DESCRIPTION

The `GXJobFormatModeQuery` function obtains information from a printer driver that relates to the printer driver's preferred mode. The kinds of queries that can be specified in the `aQueryType` parameter depend on the printer driver. The format and direction of the data transfer depends on the kind of query.

QuickDraw GX defines query types for use with printer drivers whose preferred mode is `gxTextJobFormatMode`:

Constant	Value	Explanation
<code>gxGetJobFormatLineConstraintQuery</code>	0	Used to determine line constraint characteristics
<code>gxGetJobFormatFontsQuery</code>	1	Used to determine font information
<code>gxGetJobFormatFontCommonStylesQuery</code>	2	Used to determine style names
<code>gxGetJobFormatFontConstraintQuery</code>	3	Used to determine font positioning constraints
<code>gxSetStyleJobFormatCommonStyleQuery</code>	4	Used to set style names

**RESULT CODES**

`gxSegmentLoadFailedErr`      A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

For more information the kinds of queries and the format of data returned, see “Formatting for Text Job Format Mode Printing” on page 4-36.

**Color Profile Functions**

---

QuickDraw GX allows you to find and set the color profiles that are used for color matching. Color matching and the ColorSync Manager are described in *Inside Macintosh: Advanced Color Imaging*.

**GXFindPrinterProfile**

---

You can use the `GXFindPrinterProfile` function to determine the color profile used by an output printer.

```
OSErr GXFindPrinterProfile (gxPrinter thePrinter,
                           void *searchData, long index,
                           gxColorProfile *returnedProfile, long *numProfiles);
```

`thePrinter`

A reference to the printer object.

`searchData`

A pointer to a block of data that is assumed to be a ColorSync searching block of type `CMProfileSearchRecord`. If this value is not `nil`, then the value of the `index` parameter must not be 0 if you want the search to take place.

If this value is `nil`, the value of the `index` parameter defines which profile is returned.

`index`

The index of the profile to return. If the value is 0, then the current profile is returned in the `returnedProfile` parameter.

If the value of this parameter is not 0, then the behavior this function depends on the value of the `searchData` parameter. If `index` is not 0 and `searchData` is `nil`, the indexed profile is returned in the `returnedProfile` parameter. If `index` is not 0 and `searchData` is not `nil`, then the printer profiles are searched.

## Advanced Printing Features

`returnedProfile`

On return, a list of references to color profiles matching the criteria specified by the `searchData` and `index` parameters. If no color profiles are found, this parameter is `nil` upon return.

`numProfiles`

On return, the number of profiles that were found.

*function result* An error code. The value `noErr` indicates that the operation was successful.

## DESCRIPTION

The `GXFindPrinterProfile` function searches for a color profile that matches the specifications in the `searchData` and `index` parameters.

## RESULT CODES

`gxSegmentLoadFailedErr` A required code segment could not be found, or there was not enough memory to load it.

## SEE ALSO

The `gxFindPrinterProfile` message that determines which profiles are returned is described in *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Color matching, color profiles, the `CMProfileSearchRecord` structure, and color profile resources are described in *Inside Macintosh: Advanced Color Imaging*.

## GXFindFormatProfile

---

You can use the `GXFindFormatProfile` function to determine color-matching information for a specific format object. This function is similar to the `GXFindPrinterProfile` function (described in the previous section), except that it finds a color profile that is associated with a format object rather than a printer object.

```
OSErr GXFindFormatProfile (gxFormat theFormat,
                           void *searchData, long index,
                           gxColorProfile *returnedProfile, long *numProfiles);
```

`theFormat` A reference to the format object.

## Advanced Printing Features

<code>searchData</code>	<p>A pointer to a block of data that is assumed to be a ColorSync searching block of type <code>CMProfileSearchRecord</code>. If this value is not <code>nil</code>, then the value of the <code>index</code> parameter must not be 0 if you want the search to take place.</p> <p>If this value is <code>nil</code>, the value of the <code>index</code> parameter defines which profile is returned.</p>
<code>index</code>	<p>The index of the profile to return. If the value is 0, then the current profile is returned in the <code>returnedProfile</code> parameter.</p> <p>If the value of this parameter is not 0, then the behavior this function depends on the value of the <code>searchData</code> parameter. If <code>index</code> is not 0 and <code>searchData</code> is <code>nil</code>, the indexed profile is returned in the <code>returnedProfile</code> parameter. If <code>index</code> is not 0 and <code>searchData</code> is not <code>nil</code>, then the printer profiles are searched.</p>
<code>returnedProfile</code>	<p>On return, a list of references to color profiles matching the criteria specified by the <code>searchData</code> and <code>index</code> parameters. If no color profiles are found, this parameter returns <code>nil</code>.</p>
<code>numProfiles</code>	<p>On return, the number of profiles that were found.</p>
<i>function result</i>	<p>An error code. The value <code>noErr</code> indicates that the operation was successful.</p>

## DESCRIPTION

The `GXFindFormatProfile` function searches for a color profile that matches the specifications in the `searchData` and `index` parameters.

## RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

## SEE ALSO

The `gxFindFormatProfile` message that determines which profiles are returned is described in *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Color matching, color profiles, the `CMProfileSearchRecord` structure, and color profile resources are described in *Inside Macintosh: Advanced Color Imaging*.

## GXSetPrinterProfile

---

You can call the `GXSetPrinterProfile` function to change the current color profile for a printer.

```
OSErr GXSetPrinterProfile (gxPrinter thePrinter,
                           gxColorProfile oldProfile, gxColorProfile newProfile);
```

`thePrinter`

A reference to the printer object.

`oldProfile`

A reference to the profile that has been associated with the printer object.

`newProfile`

A reference to the profile to add to the list of profiles for a printer object.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

You can call `GXSetPrinterProfile` to change the current profile for a printer, to replace an existing profile that is associated with the printer object, or to remove a profile from the list of color profiles that are associated with the printer object.

A printer driver or printing extension defines the values of the `oldProfile` and `newProfile` parameters that determine what happens in response to this message. Table 4-6 shows an example.

**Table 4-6** The actions of the `GXSetPrinterProfile` function

Value of <code>oldProfile</code>	Value of <code>newProfile</code>	Action taken
<code>nil</code>	<code>nil</code>	None
Valid	<code>nil</code>	<code>oldProfile</code> is deleted from the list of profiles associated with the printer object.
<code>nil</code>	Valid	<code>newProfile</code> is added to the list of profiles for the printer object and becomes the current profile.
Valid	Valid	<code>oldProfile</code> is deleted from the list of profiles, <code>newProfile</code> is added, and <code>newProfile</code> becomes the current profile for the printer object.

**RESULT CODES**

`gxSegmentLoadFailedErr` A required code segment could not be found, or there was not enough memory to load it.

**SEE ALSO**

The `gxSetPrinterProfile` message is described in *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Color matching, color profiles, and color profile resources are described in *Inside Macintosh: Advanced Color Imaging*.

**GXSetFormatProfile**

---

You can use the `GXSetFormatProfile` function to change the current color profile for a format object.

```
OSErr GXSetFormatProfile (gxFormat theFormat,
                          gxColorProfile oldProfile, gxColorProfile newProfile);
```

`theFormat` A reference to the format object.

`oldProfile` A reference to the profile that has been associated with the format object.

`newProfile` A reference to the profile to add to the list of profiles for a format object.

*function result* An error code. The value `noErr` indicates that the operation was successful.

**DESCRIPTION**

You can call the `GXSetFormatProfile` function to change the current profile for a format object, to replace an existing profile that is associated with the format object, or to remove a profile from the list of color profiles that are associated with the format object.

A printer driver or printing extension defines the values of the `oldProfile` and `newProfile` parameters that determine what happens in response to this message. Table 4-7 shows an example.

**Table 4-7** The actions of the `GXSetFormatProfile` function

Value of <code>oldProfile</code>	Value of <code>newProfile</code>	Action taken
<code>nil</code>	<code>nil</code>	None
Valid	<code>nil</code>	<code>oldProfile</code> is deleted from the list of profiles associated with the format object.
<code>nil</code>	Valid	<code>newProfile</code> is added to the list of profiles for the format object and becomes the current profile.
Valid	Valid	<code>oldProfile</code> is deleted from the list of profiles, <code>newProfile</code> is added, and <code>newProfile</code> becomes the current profile for the format object.

#### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
-------------------------------------	--

#### SEE ALSO

The `gxSetFormatProfile` message is described in *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

Color matching, color profiles, and color profile resources are described in *Inside Macintosh: Advanced Color Imaging*.

## Idle Job Function

---

You can call the `GXIdleJob` function to allow other applications time to execute while your application is spooling.

## GXIdleJob

---

You can use the `GXIdleJob` function to release time to other processes while your application is performing a computationally intensive task.

```
void GXIdleJob (gxJob aJob);
```

`aJob`                      A reference to a job object.

### DESCRIPTION

The `GXIdleJob` function tells QuickDraw GX to release time to other processes that are currently active. If your application is performing a computationally intensive process that can potentially lock other processes out for an extended period of time, you need to periodically call this function.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

## Application-Defined Functions

---

The following sections describe the application-defined functions for preventing the display of the Printing Status dialog box, for manipulating the view devices associated with a printer object, and for manipulating the paper types associated with a job object.

### Message Override Function for the Printing Status Dialog Box

---

You can call the `GXInstallApplicationOverride` function to install an override function for the `gxJobStatus` message to prevent the Printing Status dialog box from being displayed while spooling.



## GXJobStatus

---

QuickDraw GX sends the `gxJobStatus` message to display the current status of a print job during spooling and despooling. You can install an override function for the `gxJobStatus` message to prevent the display of status information during spooling. Your override function must match the following formal declaration:

```
OSErr GXJobStatus (gxStatusRecord *aStatusRecord);
```

`aStatusRecord`

A pointer to a status structure.

*function result* An error code. The value `noErr` indicates that the operation was successful.

### DESCRIPTION

QuickDraw GX sends the `gxJobStatus` message when a printing extension or printer driver calls the `GXReportStatus` function. This is not under the application's control.

The default implementation of this message displays the status in the desktop printer window. To prevent the display of the Printing Status dialog box, your override function should return `noErr` as its only action.

### SPECIAL CONSIDERATIONS

You never send the `gxJobStatus` message yourself.

You must forward the `gxJobStatus` message to other message handlers.

### RESULT CODES

<code>gxSegmentLoadFailedErr</code>	A required code segment could not be found, or there was not enough memory to load it.
<code>gxPrUserAbortErr</code>	The user has canceled printing.

### SEE ALSO

The status structure is described in the section “The Status Structure” on page 4-42.

For more information about status information, see the printing functions chapter of *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

## Looping Through a Printer's View Devices

---

The application-defined function called by the `GXForEachPrinterViewDeviceDo` function takes two parameters: the view device object associated with a particular printer object, and a pointer to a reference constant in which you specify data passed into the application-defined function. For example, this is how you should declare the application-function if you were to name it `MyViewDeviceFunction`:

```
gxLoopStatus MyViewDeviceFunction(gxViewDevice aViewDevice,
                                   void *refCon);
```

`aViewDevice`

A reference to the current view device. This is provided by QuickDraw GX when the `MyViewDeviceFunction` function is called.

`refCon`

A pointer to a reference constant.

*function result* A Boolean to indicate whether looping should stop.

### DESCRIPTION

When you use the `GXForEachPrinterViewDeviceDo` function, QuickDraw GX calls the application-defined function for each view device object referenced by the specified printer object until the application-defined function returns `gxStopLooping` or there are no more view devices in the list. If you want the `GXForEachPrinterViewDeviceDo` function to continue with the next view device, return `gxKeepLooping` from the application-defined function.

## Looping Through a Job's Paper Types

---

The application-defined function called by the `GXForEachJobPaperTypeDo` function takes two parameters: the view device object associated with a particular printer object, and a pointer to a reference constant in which you specify data passed into the application-defined function. For example, this is how you should declare the application-function if you were to name it `MyPaperTypeFunction`:

```
gxLoopStatus MyPaperTypeFunction(gxPaperType aPaperType,
                                   void *refCon);
```

`aPaperType`

A reference to the current paper type. This is provided by QuickDraw GX when the `MyPaperTypeFunction` function is called.

`refCon`

A pointer to a reference constant.

*function result* A Boolean to indicate whether looping should stop.

**DESCRIPTION**

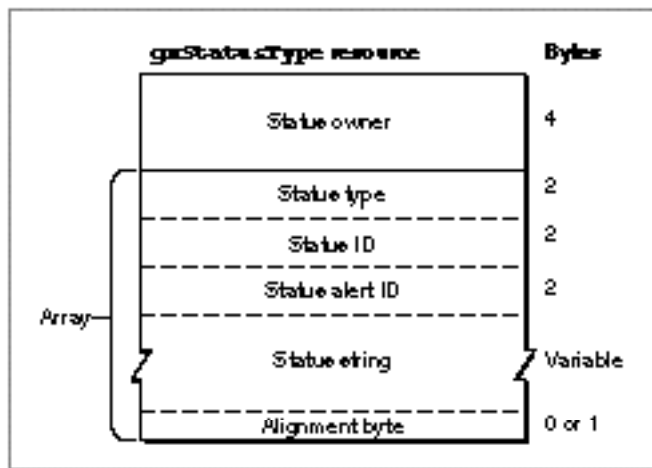
When you use the `GXForEachJobPaperTypeDo` function, QuickDraw GX calls the application-defined function for each paper-type object referenced by the specified job object until the application-defined function returns `gxStopLooping` or there are no more paper types in the list. If you want the `GXForEachJobPaperTypeDo` function to continue with the next paper type, return `gxKeepLooping` from the application-defined function.

## The Status Resource

---

You need to include a status resource, of type `gxStatusType`, to define the status messages that are displayed during the printing process. Figure 4-3 shows the structure of the status resource.

**Figure 4-3** The status resource



The status resource contains a count of the status entries and an array of status definitions.

- n Status owner. The signature of the printing extension or printer driver to which this status resource belongs.

Each status definition contains four values:

- n Status type. The kind of status message that this is. The status type constants are shown in Table 4-8.
- n Status ID. The ID of this status message within the status resource. You typically assign sequential numbers to the status messages within each status resource, as shown in the example at the end of this section.

## Advanced Printing Features

- n **Status alert ID.** The ID of the printing alert associated with this status message. Use the ID 0 to indicate that this status message does not require a printing alert.
- n **Status string.** The status message string to display to the user.

Most of the status types produce side effects. For example, if you send a status message with status type `gxSpoolingPageStatus`, the page count is incremented in the spooling status that is displayed on the user's screen. Table 4-8 shows the status type constants and the side effects associated with each.

**Table 4-8** Status types

Constant	Value	Explanation of side effects
<code>gxNonFatalError</code>	1	Affects the icon that is displayed during spooling
<code>gxFatalError</code>	2	Displays a printing alert during spooling
<code>gxPrinterReady</code>	3	Signals that alert mode is done
<code>gxUserAttention</code>	4	Signals initiation of a modal alert
<code>gxUserAlert</code>	5	Signals initiation of a printing alert
<code>gxPageTransmission</code>	6	Signals that a page has been sent to the printer and increments the printed page count
<code>gxOpenConnectionStatus</code>	7	Signals that animation of the printer icon is to begin
<code>gxInformationalStatus</code>	8	Displays an informational status message and continues
<code>gxSpoolingPageStatus</code>	9	Signals that a page has been spooled and increments the spooled page count
<code>gxEndStatus</code>	10	Signals the end of spooling
<code>gxPercentageStatus</code>	11	Signals the percentage of the current print job that is currently complete

## Summary of Advanced Printing Features

---

### Constants and Data Types for Advanced Printing Features

---

#### Job Format Modes

```
typedef OSType gxJobFormatMode;

enum {
    /* job format modes */
    gxGraphicsJobFormatMode    = (gxJobFormatMode) 'grph', /* graphics mode */
    gxTextJobFormatMode        = (gxJobFormatMode) 'text', /* text mode */
    gxPostscriptJobFormatMode  = (gxJobFormatMode) 'post' /* format mode */
};

struct gxJobFormatModeTable {
    long          numModes;      /* number of modes */
    gxJobFormatMode modes[1];    /* any number of modes */
};
```

#### Text Job Format (Direct) Mode

```
typedef long gxQueryType; /* a query type */

enum {
    /* query types */
    gxGetJobFormatLineConstraintQuery = (gxQueryType) 0, /* line */
                                                    /* constraint */
    gxGetJobFormatFontsQuery          = (gxQueryType) 1, /* fonts */
    gxGetJobFormatFontCommonStylesQuery = (gxQueryType) 2, /* font common */
                                                    /* style */
    gxGetJobFormatFontConstraintQuery = (gxQueryType) 3, /* font */
                                                    /* constraint */
    gxSetStyleJobFormatCommonStyleQuery = (gxQueryType) 4 /* common */
                                                    /* style */
};

enum { gxConstraintRange = -1 };
```

## Advanced Printing Features

```

struct gxPositionConstraintTable {
    gxPoint    phase;           /* the phase */
    gxPoint    offset;          /* the offset */
    long       numSizes;         /* the number of constraint sizes */
    Fixed      sizes[1];        /* any number of constraint sizes */
};

```

```

struct gxStyleNameTable {
    long       numStyleNames;    /* number of style names */
    Str255     styleNames[1];    /* any number of style names */
};

```

```

struct gxFontTable {
    long       numFonts;         /* number of font references */
    gxFont     fonts[1];        /* any number of font references */
};

```

**The Status Structure**

```

enum {
    /*status type IDs*/
    gxNonFatalError      = 1,    /* affects icon on spooling dialog box */
    gxFatalError          = 2,    /* sends up user alert on spooling */
                                /* dialog box */
    gxPrinterReady        = 3,    /* signals QuickDraw GX to leave alert */
                                /* mode */
    gxUserAttention        = 4,    /* signals initiation of a modal alert */
    gxUserAlert           = 5,    /* signals initiation of a movable */
                                /* modal alert */
    gxPageTransmission    = 6,    /* signals page sent to printer, */
                                /* increments page count in strings to */
                                /* user */
    gxOpenConnectionStatus = 7,    /* signals QuickDraw GX to begin */
                                /* animation of printer icon */
    gxInformationalStatus  = 8,    /* default status type, no effects */
    gxSpoolingPageStatus   = 9,    /* signals page spooled, increments */
                                /* page count in spooling dialog box */
    gxEndStatus           = 10,    /* signals end of spooling */
    gxPercentageStatus     = 11    /* signals to QuickDraw GX the amount */
                                /* of the print job which is currently */
                                /* complete */
};

```

## Advanced Printing Features

```

/* status structure */
struct gxStatusRecord {
    unsigned short statusType;    /* the type of status */
    unsigned short statusId;      /* specific status ID */
    unsigned short statusAlertId; /* printing alert ID for status */
    Signature      statusOwner;   /* status owner signature */
    short          statResId;      /* resource ID for 'stat' resource */
    short          statResIndex;   /* index into 'stat' resource */
    short          dialogResult;   /* ID of button selected to
                                   dismiss the printing alert box */
    unsigned short bufferLen;      /* # of bytes in status buffer */
    char           statusBuffer[1]; /* user response from alert */
};

```

**Pen Tables for Vector Devices**

```
enum { gxPenTableTag = 'pent' };
```

```
enum {
    /* pen widths */
    gxDeviceUnits = 0,          /* device-specific units */
    gxMmUnits = 1,              /* millimeters */
    gxInchesUnits = 2           /* inches */
};

```

```

/* pen constants */
enum { gxPenNotLoaded = -1};

```

```

/* pen table entry structure */
struct gxPenTableEntry {
    Str31      penName;          /* name of the pen */
    gxColor    penColor;         /* color that is part of the */
                                /* color set */
    fixed      penThickness;     /* size of the pen */
    short      penUnits;         /* specifies units in which pen */
                                /* thickness is defined */
    short      penPosition;      /* pen position in the carousel */
};

```

```
struct gxPenTable {
    short          numPens;      /* number of pen entries in */
                                /* the following array */
    gxPenTableEntry pens[1];     /* array of pen entries */
};
```

## Constants and Data Types for Synonyms

---

### General-Purpose PostScript Operator Synonym

```
#define gxPostScriptTag    0x706f7374      /* 'post' synonym */
```

### PostScript Control Information Synonym

```
#define gxPostControlTag    0x70736374      /* 'psct' synonym */
```

```
enum gxPsStateFlags {
    gxNoSave = 1      /* don't do save-restore around PostScript data */
};
```

```
struct gxPostControl {
    long flags;          /* PostScript state flags */
};
```

### Dash Synonym

```
#define gxDashSynonymTag 0x73647368 /* 'sdsh' synonym */
```

```
struct gxDashSynonym {
    long    size;          /* number of elements in array */
    fixed    dashLength[gxAnyNumber]; /* array of dash lengths */
};
```

### Halftone Synonym

```
enum { gxFormatHalftoneTag = 'half' };
```

```
struct gxFormatHalftoneInfo{
    long numHalftones;      /* how many halftones */
    gxHalftone halftones[1]; /* any number of halftones */
};
```



## Advanced Printing Features

```

struct gxHalftone{
    fixed      angle;          /* direction of halftone */
    fixed      frequency;      /* cells per inch */
    gxDotType   method;        /* kind of pattern */
    gxTintType  tinting;        /* tint calculation method */
    gxColor     dotColor;       /* color of foreground */
    gxColor     backgroundColor; /* color of background */
    gxColorSpace tintSpace;     /* color space for tint */
};

```

**Line Cap Synonym**

```

#define gxLineCapSynonymTag 0x6c636170 /* 'lcap' synonym */

```

```

enum gxLineCaps {
    gxButtCap      = 0,          /* square butt cap */
    gxRoundCap     = 1,          /* round cap */
    gxSquareCap    = 2,          /* square cap */
    gxTriangleCap  = 3           /* triangle cap */
};

```

```

typedef long  gxLineCapSynonym;      /* line cap type */

```

**Pattern Synonym**

```

#define gxPatternSynonymTag 0x70747265 /* 'ptrn' synonym */

```

```

enum gxPatterns {
    gxHatch        = 0,          /* hatch pattern */
    gxCrossHatch   = 1           /* crosshatch pattern */
;

```

```

struct gxPatternSynonym {
    long  patternType;          /* one of the patterns: hatch or crosshatch */
    fixed  angle;               /* angle at which pattern is drawn */
    fixed  spacing;              /* distance between two parallel pattern lines */
    fixed  thickness;            /* thickness of the pattern */
    gxPoint anchorPoint;        /* point with respect to the pattern position */
                                   /* calculated */
};

```

**Cubic Synonym**

```
#define gxCubicSynonymTag 0x63756278    /* 'cubx' synonym */

enum gxCubicSynonym {
    gxIgnoreFlag = 0x0000,    /* ignore this word, get next one */
    gxLineToFlag = 0x0001,    /* draw a line to point following this */
                                /* flag */
    gxCurveToFlag = 0x0002,    /* draw a curve through the 3 points */
                                /* following this flag */
    gxMoveToFlag = 0x0003,    /* start a new contour at the point */
                                /* following this flag */
    gxClosePathFlag = 0x0004    /* close the contour */
};

#define gxCubicInstructionMask 0x000F    /* low 4 bits are point */
                                         /* instructions */

typedef short gxCubicSynonymFlags;
/* low 8 bits are instruction (moveto, lineto, curveto, closepath) */
```

**QuickDraw Picture Synonym**

```
struct gxQuickDrawPict {
    gxTranslationOptions    options;    /* translator options */
    Rect                    srcRect;    /* QuickDraw source Rect */
    Point                   styleStretch; /* the scale factor */
    unsigned long           dataLength; /* length of picture data */
    struct gxBitmapDataSourceAlias alias;    /* alias to QuickDraw data */
};
```

**Functions**

---

**Working With Advanced Job Object Functions**

```
void GXSelectJobFormattingPrinter
    (gxJob aJob, Str31 printerName);

gxPrinter GXGetJobFormattingPrinter
    (gxJob aJob);

gxPrinter GXGetJobOutputPrinter
    (gxJob aJob);

void* GXGetJobRefcon
    (gxJob aJob);
```

## Advanced Printing Features

```
void GXSetJobRefcon      (gxJob aJob, void *refCon);
gxJob GXCopyJob          (gxJob srcJob, gxJob dstJob);
```

**Manipulating Printer Objects**

```
gxPrinter GXGetJobPrinter (gxJob aJob);
gxJob GXGetPrinterJob     (gxPrinter aPrinter);
void GXForEachPrinterViewDeviceDo
    (gxPrinter aPrinter,
     gxViewDeviceProc aViewDeviceProc,
     void *refCon);
long GXCountPrinterViewDevices
    (gxPrinter aPrinter);
gxViewDevice GXGetPrinterViewDevice
    (gxPrinter aPrinter, long whichViewDevice);
void GXSelectPrinterViewDevice
    (gxPrinter aPrinter, long whichViewDevice);
void GXGetPrinterDriverName (gxPrinter aPrinter, Str31 name);
void GXGetPrinterName       (gxPrinter aPrinter, Str31 name);
OSType GXGetPrinterDriverType
    (gxPrinter aPrinter);
OSType GXGetPrinterType     (gxPrinter aPrinter);
```

**Working With QuickDraw GX Print Files**

```
gxPrintFile GXOpenPrintFile (gxJob aPrintFileJob, FSSpecPtr pFileSpec,
                             char permission);
void GXClosePrintFile       (gxPrintFile aPrintFile);
gxJob GXGetPrintFileJob     (gxPrintFile aPrintFile);
long GXCountPrintFilePages  (gxPrintFile aPrintFile);
void GXReadPrintFilePage    (gxPrintFile aPrintFile, long pageNumber,
                             long numViewPorts, gxViewPort *viewPortList,
                             gxFormat *pageFormat, gxShape *pageShape);
void GXReplacePrintFilePage (gxPrintFile aPrintFile,
                             long pageNumber, gxFormat pageFormat,
                             gxShape pageShape);
void GXInsertPrintFilePage  (gxPrintFile aPrintFile, long atPageNumber,
                             gxFormat pageFormat, gxShape pageShape);
void GXDeletePrintFilePageRange
    (gxPrintFile aPrintFile,
     long fromPageNumber,
     long toPageNumber);
void GXSavePrintFile        (gxPrintFile aPrintFile, FSSpec *pFileSpec);
```

**Working With Paper Types**

```

gxPaperType GXNewPaperType (gxJob aJob, Str31 name,
                             gxRectangle *pageSize, gxRectangle *paperSize);

void GXDisposePaperType (gxPaperType aPaperType);

gxPaperType GXGetNewPaperType
                             (gxJob aJob, short resID);

gxPaperType GXGetJobPaperType
                             (gxJob aJob, long whichPaperType,
                              Boolean forFormatDevice,
                              gxPaperType aPaperType);

long GXCountJobPaperTypes (gxJob aJob, Boolean forFormatDevice);

gxPaperType GXCopyPaperType (gxPaperType srcPaperType,
                              gxPaperType dstPaperType);

void GXGetPaperTypeName (gxPaperType aPaperType,
                          Str31 name);

void GXGetPaperTypeDimensions
                             (gxPaperType aPaperType,
                              gxRectangle *aPageSize,
                              gxRectangle *aPaperSize);

gxJob GXGetPaperTypeJob (gxPaperType aPaperType);

void GXForEachJobPaperTypeDo
                             (gxJob aJob, gxPaperTypeProc aPaperTypeProc,
                              void *refCon, Boolean forFormattingPrinter);

```

**Formatting for Specific Devices**

```

void GXSetAvailableJobFormatModes
                             (gxJob aJob,
                              JobFormatModeTableHdl
                              aJobFormatModeTableHdl);

gxJobFormatMode GXGetPreferredJobFormatMode
                             (gxJob aJob,
                              Boolean *directOnly);

gxJobFormatMode GXGetJobFormatMode
                             (gxJob aJob);

void GXSetJobFormatMode (gxJob aJob, gxJobFormatMode aMode);

void GXJobFormatModeQuery (gxJob aJob, gxQueryType aQueryType,
                           void *srcData, void *dstData);

```

**Color Profile Functions**

```

OSErr GXFindPrinterProfile (gxPrinter thePrinter, void *searchData,
                           long index, gxColorProfile *returnedProfile,
                           long *numProfiles);

OSErr GXFindFormatProfile (gxFormat theFormat, void *searchData,
                           long index, gxColorProfile *returnedProfile,
                           long *numProfiles);

OSErr GXSetPrinterProfile (gxPrinter thePrinter,
                           gxColorProfile oldProfile,
                           gxColorProfile newProfile);

OSErr GXSetFormatProfile (gxFormat theFormat,
                           gxColorProfile oldProfile,
                           gxColorProfile newProfile);

```

**Idle Job Function**

```
void GXIdleJob (gxJob aJob);
```

**Application-Defined Functions**

```

OSErr GXJobStatus (gxStatusRecord *aStatusRecord);

gxLoopStatus MyViewDeviceFunction
               (gxViewDevice aViewDevice,
               void *refCon);

gxLoopStatus MyPaperTypeFunction
               (gxPaperType aPaperType,
               void *refCon);

```



# Glossary

---

**application phase** In QuickDraw GX printing, the phase when the application calls QuickDraw GX and interacts with the user by displaying dialog boxes to establish printing parameters, such as page orientation and paper type.

**CMYK color space** A color space whose four components measure the cyan, magenta, yellow, and black elements of a color. Used mostly for printing.

**collection object** An extensible object, managed by the Collection Manager, that is used to hold any kind of information. See **job collection**, **format collection**, **paper-type collection**.

**color matching** A method of accurately converting colors in one color space to another color space, or from display on one device to display on another device.

**color profile** A QuickDraw GX object associated with a transfer mode, color, or bitmap data structure and used for color matching. A color profile usually describes the color response curve of a display device in terms of an objective standard.

**color space** A specification of a particular method for color representation, such as RGB, CMYK, or gray space.

**ColorSync Utilities** A part of Macintosh system software that manages color matching, color profiles, and the drawing of matched colors. QuickDraw GX color profile objects contain ColorSync color profiles, and QuickDraw GX uses the ColorSync Utilities to perform its color matching.

**despool** To open a print file and send its data to a device for printing. See **spooling phase**.

**direct mode** See **job format mode**.

**desktop printer** The representation of a QuickDraw GX printer as an icon on the user's desktop.

**device communications phase** In QuickDraw GX printing, the phase when the data that represents the rendered form of each page is sent to the output device.

**extended item list** A resource that extends an item list ( 'DITL' ) resource by responding automatically to items when they are manipulated by the user.

**flatten** To convert the private, object-based description of an object or set of objects into a public-format data stream suitable for file or clipboard storage. Used when saving a print job. Compare **unflatten**.

**formatting printer** The printer for which a document's format is retained. See also **output printer**.

**format collection** A collection of items that are relevant to a format but are not required to define a format. See also **collection object**.

**format object** An object that represents how pages of a document are to be formatted, including scaling, orientation, and paper type. It allows a form to be associated with a format. See also **paper-type object**, **form**.

**form** A property of a format object that allows a picture shape to be printed as a backdrop to the contents of the page. A form can optionally include a mask shape that defines areas that are not printed.

**forward** To pass a message on to the next message handler in a message chain. See also **message chain**, **message handler**, **override**.

**gamut** The limits of the colors that a device can produce. Different devices have different gamuts, so color matching is necessary when converting colors from one device to another.

**grayscale** Consisting entirely of shades of gray.

**gray space** A color space whose single component is the lightness or brightness of a color.

**halftone** A QuickDraw GX data structure that specifies a pattern and a set of colors. A halftone is used to achieve a greater range of colors that may be otherwise available.

**imaging phase** In QuickDraw GX printing, the phase when each previously spooled page is rendered into a form that can be printed on the output device.

**job collection** A collection of items that are relevant for a print job but not required to define a print job. See also **collection object**.

**job format mode** A mode of printing, either graphics (the QuickDraw GX default), text-only, or PostScript-only. The text and PostScript modes are sometimes called direct-mode printing; used to trade off the ability to redirect output to another printer for faster output on a specific printer.

**job object** An object that represents the parameters associated with printing, such as the printer and page range. These parameters specify a “print job.”

**mapping** A  $3 \times 3$  matrix—a property of a format object that specifies scaling and orientation.

**message** A notice sent by one message handler to another that a certain condition has arisen or that a certain task needs to be accomplished. See also **printing message**.

**message chain** A hierarchy of message handlers eligible to receive and respond to messages.

**message handler** A recipient of messages. In QuickDraw GX printing, applications, printing extensions, printer drivers, and QuickDraw GX are all message handlers, which are part of a message chain.

**message override** The response, by a message handler, of intercepting a message and taking some action. The response to a message is performed by an override function. See also **override function**.

**message-passing architecture** A software system driven by messages that are sent in response to certain conditions or events. The messages activate message handlers, which take action in response to the messages. QuickDraw GX printing uses a message-passing architecture.

**output printer** The printer to which a document is sent to be printed. If the document’s formatting printer is different than the output printer, the print file reflects the output printer’s formatting; however, the document itself retains its original format. See also **formatting printer**.

**override** (n.) See **message override** and **override function**. (v.) To intercept a message and take action on it.

**object** A private QuickDraw GX data structure. An object has specific properties and is accessed through a reference.

**override function** The code, defined in a message handler, that responds to a message. See also **message override**.

**owner** A variable, structure, or QuickDraw GX object that references an object. Many objects can be referenced by more than one variable and can thus have multiple owners.

**panel** A subset of a dialog box used to display and collect related pieces of information. An expanded dialog box may contain one or more panels, each of which is named and associated with an icon. A panel is defined by a panel resource.

**paper-type collection** A collection of items that are relevant to a kind of paper but are not required to define a paper type. See also **collection object**.

**paper-type object** A paper-type object represents the paper for which a page is formatted. It specifies the name of the paper, the size, and the printable area. See also **format object**.

**partial override** An implementation of a printing message override that forwards the message to other message handlers. You typically forward the message at the beginning or end of your override function.



**picture shape** A shape type that represents a collection of other shapes.

**portable digital document (PDD)** A print file that can be viewed without the application or fonts that created it. It is created by printing with the PDD Maker GX printer driver. See also **print file**.

**printer** See **desktop printer**, **formatting printer**, **output printer**.

**printer driver** A program that converts data that is sent by an application program into data and control sequences intended for a specific output device.

**printer object** An object that represents the characteristics of a printer, such as its color space and resolution. The output printer and formatting printer are represented by printer objects.

**print dialog box** A dialog box provided by QuickDraw GX that is both movable and modal. Most print dialog boxes have both a normal and an expanded version. The application can customize print dialog boxes by adding panels. See also **panel**.

**print file** A document or data that has been spooled by actually printing the file. See also **portable digital document**.

**print file object** A representation of a print file, which allows an application to access the contents of the file.

**print job** See **job object**.

**printing extension** An add-on software module that allows you to extend printing functionality provided by applications and printer drivers.

**printing message** A notice that QuickDraw GX sends to the message handlers in a message chain that a certain printing-related condition has arisen or that a certain printing-related task needs to be accomplished. See also **message chain**, **message handler**.

**printing message override** See **message override**.

**property** An item or set of data in a QuickDraw GX object. A property of an object is analogous to a field (or member) of a data structure; however, a field is accessed through its name, whereas a property is accessed through a function.

**reference** A long-word value, neither a pointer nor a handle, through which an application accesses a QuickDraw GX object. References are created by QuickDraw GX and passed to applications.

**scale** To proportionally enlarge or shrink.

**shape** (1) A graphic or typographic item (such as a geometric shape, a bitmap, or a line of text) created and drawn with QuickDraw GX. (2) A set of QuickDraw GX objects that, taken together, describe the type and characteristics of such a graphic or typographic item. A shape consists of a shape object, a style object, an ink object, and a transform object.

**shape object** A QuickDraw GX object that, along with several other objects, describes a QuickDraw GX shape. A shape object specifies the fundamental type and contents of a shape.

**spooling phase** In QuickDraw GX printing, the phase when the application sends the document pages to disk, in preparation for printing. The printer driver stores printable output in a file from which it is subsequently despoiled, rendered, and sent to the output device. See also **despool**.

**synonym** A particular kind of tag object, used by QuickDraw GX to provide an alternate representation of an object for printing. The synonym specifies data, such as alternative PostScript operators, for the printer driver to use instead of the instructions that QuickDraw GX generates.

**tag list** A property of many QuickDraw GX objects. It is an array of references to tag objects associated with the object.

**tag object** A QuickDraw GX object whose purpose, structure, and content are entirely controlled by the application creating it. Tag objects exist to allow custom information and behavior to be attached to standard QuickDraw

## G L O S S A R Y

**GX objects.** Tag objects are classified by tag type; objects reference their tag objects through a tag list.

**tag type** A longword data type (equivalent to `OSType`) that can be represented by four 1-byte characters, such as 'appl'. Tag types specify the formats of tag objects, such as synonyms.

**total override** An implementation of a printing message override that does not forward the message to other message handlers.

**unflatten** To convert the public, stream-based description of an object or set of objects into the private, native QuickDraw GX object-based format. Used when retrieving a print job. Compare **flatten**.

# Index

---

## A

---

advanced printing features 1-30, 4-5 to 4-102  
application phase of printing 1-4  
attribute bit masks, for printing-related collections 3-9,  
3-76

---

## B

---

base information  
as paper-type collection item 3-14  
defined 3-94

---

## C

---

caps, synonym 4-14 to 4-15, 4-47  
collation information  
as job collection item 3-11  
defined 3-80  
collection items, printing-related 3-7 to 3-8  
collection objects. *See also* printing-related collections  
printing tag 3-8, 3-77  
Collection type 2-47  
color matching, for printers 4-9  
color profile objects, for printers 4-9, 4-27 to 4-29, 4-84  
to 4-89  
color spaces, printer specification 4-8, 4-27 to 4-29  
comment information  
as paper-type collection item 3-14  
defined 3-97  
compatibility of QuickDraw GX printing with  
Macintosh Printing Manager 1-30  
copies information  
as job collection item 3-11  
defined 3-81  
core printing features 1-26 to 1-28, 2-3 to 2-81  
creator information  
as paper-type collection item 3-14  
defined 3-95  
customizing printing features 1-28 to 1-29, 3-22 to  
3-27, 3-33 to 3-39, 3-66 to 3-75, 3-98 to 3-102,  
3-113 to 3-125  
Custom Page Setup dialog box 3-113

---

## D

---

dashes, synonym 4-14, 4-46  
desktop printers 1-7 to 1-8  
application support for 2-39 to 2-42  
icons for 1-9  
device communications phase of printing 1-5  
dialog boxes 1-10 to 1-13, 2-17 to 2-19  
customization 3-6 to 3-7, 3-22 to 3-27, 3-66 to 3-73,  
3-119 to 3-125  
Custom Page Setup dialog box 3-113  
displaying 2-37 to 2-39, 2-71 to 2-74, 3-23, 3-113 to  
3-121  
Edit menu structure 2-9  
extended item list resources 3-72 to 3-73, 3-128 to  
3-132  
Page Setup dialog box 2-35 to 2-37, 2-72, 3-121  
panel resources 3-24, 3-70, 3-127  
parsing responses in 3-102  
Print dialog box 1-10 to 1-12, 2-37 to 2-39, 2-73, 3-120  
Printing Status dialog box 4-91  
results defined 2-48  
setting up panels 3-67 to 3-69, 3-114  
dimensions  
as format object property 2-8  
as paper-type object property 2-9  
from format objects 2-33 to 2-34, 2-70  
from paper-type objects 4-33, 4-77  
of panels 2-7, 3-115  
direct mode 4-19 to 4-20, 4-35 to 4-36. *See also* job  
format modes  
direct-mode information  
as format collection item 3-13  
defined 3-91  
direct mode. *See* job format modes  
'DITL' resource type 3-71

---

## E

---

Edit menu structure 2-9  
defined 2-48  
error. *See also* printing errors  
as job object property 2-6  
extended item list resources 3-72 to 3-73, 3-128 to 3-132

## F

---

- file-destination information
  - as job collection item 3-11
  - defined 3-83
- file-fonts information
  - as job collection item 3-11
  - defined 3-85
- file-format information
  - as job collection item 3-11
  - defined 3-84
- file-location information
  - as job collection item 3-11
  - defined 3-84
- Finder printing support 2-39 to 2-42
- flags information
  - as paper-type collection item 3-14
  - defined 3-97
- flattening
  - job objects 2-25 to 2-28, 2-56 to 2-58, 2-77
  - print-job flattening function 2-27 to 2-28
  - print-job to handle 2-25 to 2-27
- font table 4-37, 4-41
- format collection items
  - changing 3-112
  - direct-mode information 3-13, 3-91
  - format-half-tone information 3-13, 3-92
  - horizontal page-flip information 3-13, 3-93
  - orientation information 3-13, 3-89
  - page-inversion information 3-13, 3-92
  - paper-type lock information 3-13, 3-94
  - precise-bitmap information 3-13, 3-93
  - scaling information 3-13, 3-91
  - vertical page-flip information 3-13, 3-93
- format collections 3-12 to 3-13
  - accessing 3-118
  - accessing page correspondences 3-61 to 3-66
  - as format object property 2-8
  - constants and data types for 3-89 to 3-94
  - half-tone information 3-21 to 3-22, 3-52 to 3-53
  - items in. *See* format collection items
  - mapping items 3-18 to 3-19
- format-half-tone information. *See also* halftones
  - as format collection item 3-13
  - defined 3-92
- format list, as print file object property 4-10
- format mode. *See also* job format modes
  - as job object property 2-6
- format object properties 2-7 to 2-8
  - collection 2-8
  - dimensions 2-8
  - form 2-8
  - job object 2-8
  - mapping 2-8
  - paper type 2-8
- format objects 1-17, 2-3 to 2-5, 2-7 to 2-8
  - accessing format collections 3-118
  - as job object property 2-7
  - changing collection items 3-112
  - cloning 3-44 to 3-47, 3-106
  - copying 3-54 to 3-56, 3-105
  - counting owners 3-107
  - creating 3-40 to 3-44, 3-104
  - current format mode 4-81, 4-82
  - defined 2-47
  - disposing of 3-47 to 3-50, 3-104
  - forms 3-20, 3-50 to 3-51, 3-111
  - manipulating 3-16 to 3-22
  - manipulating properties of 2-33 to 2-34, 3-103 to 3-112
  - manipulation by job objects 3-108, 3-126
  - mapping 3-18 to 3-19
  - obtaining job object from 2-33, 2-69
  - properties of. *See* format object properties
  - retrieving dimensions 2-33 to 2-34, 2-70
  - retrieving forms 3-111
  - retrieving from job object 2-69
  - retrieving mapping 3-57, 3-109
  - retrieving the paper type 3-57 to 3-59, 3-110
  - sharing 3-44 to 3-47
- format-panel information
  - as job collection item 3-12
  - defined 3-88
- formatting
  - associating pages and formats 3-61 to 3-66
  - for specific devices 4-79 to 4-84
  - page formatting 3-6 to 3-7, 3-15 to 3-22, 3-39 to 3-66
  - text mode queries 4-36 to 4-37, 4-83
- formatting printers
  - as job object property 2-6
  - changing color profiles 4-88
  - determining for job objects 4-51
  - retrieving color profiles 4-85
  - selecting 4-50
- forms 1-17
  - as format object property 2-8
  - printing 3-20
  - retrieving 3-111
  - specifying 3-111
  - using 3-50 to 3-51
- forwarding messages 3-23

## GA–GXB

---

Gestalt selectors, for QuickDraw GX printing 2-11, 2-47  
 gxBaseInfo structure 3-94

## GXC

---

GXChangedFormat function 3-112  
 GXCloneFormat function 3-46, 3-106  
 GXClosePrintFile function 4-29, 4-63  
 gxCollationInfo structure 3-80  
 gxCollectionCategory type 3-9, 3-77  
 gxCommentInfo structure 3-98  
 GXConvertPrintRecord function 2-45, 2-75  
 gxCopiesInfo structure 3-81  
 GXCopyFormat function 3-54, 3-105  
 GXCopyJob function 4-25, 4-53  
 GXCopyPaperType function 4-76  
 GXCountFormatOwners function 3-107  
 GXCountJobFormats function 3-107  
 GXCountJobPaperTypes function 4-75  
 GXCountPrinterViewDevices function 4-26, 4-57  
 GXCountPrintFilePages function 4-29, 4-30, 4-65  
 gxCreatorInfo structure 3-95  
 gxCubicSynonym enumeration 4-48  
 gxCubicSynonymFlags type 4-48

## GXD

---

gxDashSynonym structure 4-14, 4-46  
 GXDeletePrintFilePageRange function 4-69  
 gxDialogResult type 2-48  
 gxDirectModeInfo structure 3-91  
 GXDisposeFormat function 3-49, 3-54, 3-104  
 GXDisposeJob function 2-28, 2-29, 2-55  
 GXDisposePaperType function 4-72  
 GXDrawShape function 2-20, 2-22, 2-64

## GXE

---

gxEditMenuRecord structure 2-9, 2-48  
 GXEnableJobScalingPanel function 3-116  
 GXEnterGraphics function 2-11, 2-50  
 GXExitGraphics function 2-12, 2-51  
 GXExitPrinting function 2-12, 2-51  
 gxExtendedDITLType resource type 3-72 to 3-73,  
 3-128 to 3-132

## GXF

---

gxFileDestinationInfo structure 3-83  
 gxFileFontsInfo structure 3-85  
 gxFileFormatInfo structure 3-84  
 gxFileLocationInfo structure 3-84  
 GXFilterPanelEvent message 3-124  
 GXFindFormatProfile function 4-85  
 GXFindPrinterProfile function 4-84  
 GXFinishJob function 2-20, 2-65  
 GXFinishPage function 2-20, 2-22, 2-64, 2-67  
 gxFlagsInfo structure 3-97  
 GXFlattenJob function 2-25, 2-27, 2-28, 2-29, 2-57  
 GXFlattenJobToHdl function 2-25, 2-29, 2-56  
 gxFlipPageHorizontalInfo structure 3-93  
 gxFlipPageVerticalInfo structure 3-93  
 gxFontTable structure 4-37, 4-41  
 GXForEachFormatDo function 3-108  
 GXForEachJobFormatDo function 3-60  
 GXForEachJobPaperTypeDo function 4-34, 4-78  
 GXForEachPrinterViewDeviceDo function 4-56  
 GXFormatDialog function 3-113, 4-24  
 GXFormatDialog message 3-122  
 gxFormatHalftoneInfo structure 3-92, 4-15, 4-46  
 gxFormatPanelInfo structure 3-88  
 gxFormat type 2-47

## GXG

---

GXGetFormatCollection function 3-53, 3-118  
 GXGetFormatDimensions function 2-12, 2-70  
 GXGetFormatForm function 3-111  
 GXGetFormatJob function 2-33, 2-69  
 GXGetFormatMapping function 3-57, 3-109  
 GXGetFormatPaperType function 3-58, 3-110  
 GXGetJobCollection function 3-28, 3-117  
 GXGetJobError function 2-14, 2-52  
 GXGetJobFormat function 2-21, 2-69  
 GXGetJobFormatMode function 4-36, 4-81  
 GXGetJobFormattingPrinter function 4-26, 4-51  
 GXGetJobOutputPrinter function 4-22, 4-51  
 GXGetJobPageRange function 2-20, 2-62  
 GXGetJobPanelDimensions function 3-115  
 GXGetJobPaperType function 4-74  
 GXGetJobPrinter function 4-55  
 GXGetJobRefCon function 4-52  
 GXGetMessageHandlerResFile function 3-116  
 GXGetNewPaperType function 4-73  
 GXGetPaperTypeCollection function 3-118  
 GXGetPaperTypeDimensions 4-33  
 GXGetPaperTypeDimensions function 4-77

GXGetPaperTypeJob **function** 4-78  
 GXGetPaperTypeName **function** 4-32, 4-76  
 GXGetPreferredJobFormatMode **function** 4-80  
 GXGetPrinterDriverName **function** 4-22, 4-59  
 GXGetPrinterDriverType **function** 4-22, 4-60  
 GXGetPrinterJob **function** 4-55  
 GXGetPrinterName **function** 4-22, 4-59  
 GXGetPrinterType **function** 4-22, 4-61  
 GXGetPrinterViewDevice **function** 4-26, 4-57  
 GXGetPrintFileJob **function** 4-64

## GXH, GXI

---

GXHandlePanelEvent **message** 3-123  
 GXIdleJob **function** 4-90  
 GXInitPrinting **function** 2-11, 2-50  
 GXInsertPrintFilePage **function** 4-68  
 GXInstallApplicationOverride **function** 2-18, 2-54, 2-71, 3-67  
 gxInvertPageInfo **structure** 3-92

## GXJ, GXK

---

GXJobDefaultFormatDialog **function** 2-35, 2-72  
 GXJobDefaultFormatDialog **message** 3-121  
 GXJobFormatModeQuery **function** 4-36, 4-83  
 gxJobFormatModeTable **type** 4-39  
 gxJobFormatMode **type** 4-39  
 gxJobInfo **structure** 3-78  
 GXJobPrintDialog **function** 2-38, 2-73  
 GXJobPrintDialog **message** 3-120  
 GXJobStatus **message** 4-91  
 gxJob **type** 2-47

## GXL, GXM

---

gxLineCaps **enumeration** 4-47  
 gxLineCapSynonym **type** 4-14, 4-47  
 gxLoopStatus **type** 3-60, 3-76  
 gxManualFeedInfo **structure** 3-86

## GXN

---

GXNewFormat **function** 3-42, 3-54, 3-104  
 GXNewJob **function** 2-13, 2-54  
 GXNewPaperType **function** 4-32, 4-71  
 gxNormalMappingInfo **structure** 3-86

## GXO

---

GXOpenPrintFile **function** 4-29, 4-62  
 gxOrientationInfo **structure** 3-90

## GXP

---

gxPageRangeInfo **structure** 3-81  
 gxPanelEvent **enumeration** 3-99  
 gxPanelInfoRecord **structure** 3-98  
 gxPanelSetupRecord **structure** 3-101  
 gxPaperFeedInfo **structure** 3-85  
 gxPaperTypeLockInfo **structure** 3-94  
 gxPaperType **type** 2-47  
 GXParsePageRange **message** 3-125  
 gxParsePageRangeResult **type** 3-102  
 gxPatterns **enumeration** 4-48  
 gxPatternSynonym **structure** 4-17, 4-47  
 gxPenTableEntry **structure** 4-21, 4-44  
 gxPenTable **structure** 4-21, 4-44  
 gxPositionConstraintTable **structure** 4-37, 4-41  
 gxPostControl **structure** 4-14, 4-45  
 gxPreciseBitmapInfo **structure** 3-94  
 gxPrinter **type** 2-47  
 gxPrintFile **type** 2-47  
 GXPrintingEvent **message** 2-17, 2-18, 2-76  
 gxPrintingPanelKind **enumeration** 3-102  
 GXPrintPage **function** 2-20, 2-21, 2-64, 2-69  
 gxPrintPanelInfo **structure** 3-88  
 gxPrintPanelType **resource type** 3-24, 3-70, 3-127  
 gxPsStateFlags **enumeration** 4-45

## GXQ

---

gxQualityInfo **structure** 3-83  
 gxQueryType **type** 4-40  
 gxQuickDrawPict **structure** 4-18, 4-49

## GXR

---

GXReadPrintFilePage **function** 4-30, 4-65  
 GXReplacePrintFilePage **function** 4-66

## GXS

---

GXSavePrintFile **function** 4-70  
 gxScalingInfo **structure** 3-91

GXSelectJobFormattingPrinter function 4-50  
 GXSelectJobOutputPrinter function 2-40, 2-61  
 GXSelectPrinterViewDevice function 4-58  
 GXSetAvailableJobFormatModes function 4-35, 4-80  
 GXSetFormatForm function 3-50, 3-111  
 GXSetFormatProfile function 4-88  
 GXSetJobError function 2-17, 2-53  
 GXSetJobFormatMode function 4-36, 4-82  
 GXSetJobRefCon function 4-23, 4-53  
 GXSetPrinterProfile function 4-87  
 GXSetupDialogPanel function 3-68  
 GXSetupPanel function 3-114  
 gxSimplePageRangeInfo structure 3-82  
 gxSpecialMappingInfo structure 3-87  
 GXStartJob function 2-20, 2-63  
 GXStartPage function 2-20, 2-22, 2-64, 2-66, 2-69  
 gxStatusRecord structure 4-42  
 gxStatusType resource type 4-93 to 4-94  
 gxStyleNameTable structure 4-37, 4-41

## GXT

---

gxTranslatedDocumentInfo structure 3-89  
 gxTrayIndex type 3-88  
 gxTrayMappingInfo structure 3-88

## GXU–GXZ

---

GXUnflattenJobFromHdl function 2-30, 2-58  
 GXUnflattenJob function 2-29, 2-32, 2-59  
 gxUnitsInfo structure 3-96  
 GXUpdateJob function 2-42, 2-60

## H

---

halftones  
   printing with 3-21 to 3-22, 4-15 to 4-16, 4-46  
   specifying 3-52 to 3-53, 4-46  
   synonym for 4-46  
 horizontal page-flip information  
   as format collection item 3-13  
   defined 3-93

## I

---

imaging phase of printing 1-5  
 item list resource type 3-71

## J, K

---

job collection items  
   collation information 3-11, 3-80  
   copies information 3-11, 3-81  
   file-destination information 3-11, 3-83  
   file-fonts information 3-11, 3-85  
   file-format information 3-11, 3-84  
   file-location information 3-11, 3-84  
   format-panel information 3-12, 3-88  
   manual-feed information 3-11, 3-86  
   page-range information 3-11, 3-81  
   paper-feed information 3-11, 3-85  
   paper-mapping information 3-12, 3-89  
   print-job information 3-10, 3-78  
   print-panel information 3-12, 3-88  
   quality information 3-11, 3-83  
   special mapping information 3-11, 3-12, 3-87  
   standard mapping information 3-11, 3-86  
   translated-document information 3-12, 3-89  
   tray-mapping information 3-12, 3-88  
 job collections 3-10 to 3-12  
   accessing 3-117  
   as job object property 2-7  
   constants and data types for 3-78 to 3-89  
   items in. *See* job collection items  
 job format modes  
   defined 4-19 to 4-20, 4-39  
   determining preferred mode 4-80  
   retrieving current mode 4-81  
   setting current mode 4-82  
   specifying 4-80  
   text formatting 4-36 to 4-37, 4-40 to 4-41, 4-83  
   text query types defined 4-40  
   using 4-35 to 4-36  
 job object properties 2-5 to 2-7  
   collection 2-7  
   error 2-6  
   format list 2-7  
   format mode 2-6  
   formatting printer 2-6  
   output printer 2-6  
   page range 2-7  
   panel dimensions 2-7  
   paper-type list 2-7  
   reference constant 2-6

job objects 1-16 to 1-17, 2-3 to 2-5, 2-5 to 2-7  
     accessing job collections 3-117  
     as format object property 2-8  
     as paper-type object property 2-9  
     as printer object property 4-7  
     as print file object property 4-10  
     copying 4-25, 4-53  
     counting format objects 3-107  
     counting paper-types for 4-75  
     creating 2-12 to 2-14, 2-54  
     defined 2-47  
     determining page range for 2-62  
     disposing of 2-28 to 2-29, 2-55  
     error for 2-52  
     finishing a print job 2-65  
     flattening 2-25 to 2-28, 2-56 to 2-58, 2-77  
     manipulating 2-54 to 2-60  
     manipulating format objects 3-108, 3-126  
     manipulating paper types 4-34 to 4-35, 4-78, 4-92  
     manipulating properties of 4-23 to 4-24, 4-50 to 4-54  
     printing with 2-61 to 2-68  
     print record conversion 2-75  
     properties of. *See* job object properties  
     retrieving 2-29 to 2-32  
     retrieving a printer 4-55  
     retrieving format objects from 2-69, 3-59 to 3-61  
     retrieving from format objects 2-69  
     retrieving panel dimensions 3-115  
     retrieving paper-type objects 4-74  
     retrieving printer information 4-22 to 4-23  
     saving 2-24 to 2-28  
     setting error condition 2-53  
     setting error conditions for 2-17  
     unflattening 2-30 to 2-32, 2-58 to 2-60, 2-78  
     updating 2-42 to 2-44, 2-60

## L

---

line caps. *See* Caps  
 loop status information 3-76

## M, N

---

Macintosh Printing Manager  
     compatibility with QuickDraw GX 1-30, 2-75  
     printing documents created with 2-44 to 2-45  
 manual-feed information  
     as job collection item 3-11  
     defined 3-86

mappings  
     and format objects 3-18 to 3-19, 3-57, 3-109  
     as format object property 2-8  
 message chain 1-13  
 message handlers 1-13, 2-71  
 messages 1-13 to 1-15  
     forwarding 3-23  
     installing handlers for 2-71  
     printing messages 1-13  
     retrieving resources for handler 3-116  
 MyDocumentRec structure 2-10  
 MyFlattenFunction application-defined  
     function 2-27, 2-77  
 MyFormatFunction application-defined function 3-60,  
     3-126  
 MyPaperTypeFunction application-defined  
     function 4-34, 4-92  
 MyUnflattenFunction application-defined  
     function 2-32, 2-78  
 MyViewDeviceFunction application-defined  
     function 4-92

## O

---

objects. *See also* job objects; format objects; paper-type  
     objects; printer objects; print file objects  
     printing-related 1-6 to 1-7, 1-16 to 1-22  
 orientation information  
     as format collection item 3-13  
     defined 3-89  
 output printers  
     as job object property 2-6  
     changing color profiles 4-87  
     determining for job objects 4-51  
     retrieving color profiles 4-84  
     selecting 2-61  
 override functions 1-13  
     MyFormatDialogOverride application-defined  
         function 3-67  
     MyParsePageRangeOverride application-defined  
         function 3-75  
     MyPrintDialog application-defined function 3-73  
     MyPrintingEvent application-defined function 2-19  
     setting up 2-71

## P

---

page count  
     as print file object property 4-10  
     determining for print file 4-29, 4-65  
 page formatting. *See* formatting



- page-inversion information
  - as format collection item 3-13
  - defined 3-92
- page-range information 3-33 to 3-39
  - as job collection item 3-11
  - defined 3-81
  - parsing 3-73 to 3-75
- page ranges
  - as job object property 2-7
  - deleting from print files 4-69
  - determining 2-62
- Page Setup dialog box 2-35 to 2-37, 2-72, 3-121
- page size. *See also* dimensions
  - from paper type 4-33, 4-77
- panel dimensions
  - as job object property 2-7
  - determining 3-115
- panel events
  - actions 3-101
  - automated responses 3-25 to 3-27
  - constants and data types 3-99 to 3-102
  - handling 3-25 to 3-27, 3-123 to 3-125
- panel information structure 3-98
- panel resources 3-24, 3-70, 3-127
- panels 1-11 to 1-13. *See also* panel events
  - automating responses in 3-25 to 3-27
  - custom 3-22 to 3-24
  - for scaling 3-116
  - resources 3-24, 3-70, 3-127
  - retrieving dimensions of 3-115
  - setting up 3-67 to 3-69, 3-114
- panel setup information structure 3-101
- paper-feed information
  - as job collection item 3-11
  - defined 3-85
- paper-mapping information, as job collection item 3-12
- paper size. *See also* dimensions
  - from paper type 4-33, 4-77
- paper-type collection items
  - base information 3-14, 3-94
  - comment information 3-14, 3-97
  - creator information 3-14, 3-95
  - flags information 3-14, 3-97
  - units information 3-14, 3-96
- paper-type collections 3-14
  - accessing 3-118
  - as paper-type object property 2-9
  - constants and data types for 3-94 to 3-98
  - items in. *See* paper-type collection items
- paper-type lock information
  - as format collection item 3-13
  - defined 3-94
- paper-type name
  - as paper-type object property 2-9
  - determining 4-76
- paper-type object properties 2-8 to 2-9
  - collection 2-9
  - dimensions 2-9
  - job object 2-9
  - name 2-9
- paper-type objects 1-18, 2-3 to 2-5, 2-8 to 2-9
  - accessing paper-type collections 3-118
  - as format object property 2-8
  - as job object property 2-7
  - copying 4-76
  - creating 4-32, 4-71
  - creating from resources 4-73
  - defined 2-47
  - determining paper and page sizes 4-33, 4-77
  - determining the name of 4-32 to 4-33
  - disposing of 4-72
  - manipulating for a job object 4-34 to 4-35, 4-78, 4-92
  - manipulating properties of 4-71 to 4-79
  - properties of. *See* paper-type object properties
  - retrieving by format objects 3-57 to 3-59, 3-110
  - retrieving job object from 4-74, 4-78
  - retrieving the name 4-76
- parse range results enumeration 3-102
- parsing page range information 3-73 to 3-75
- partial overrides of printing messages 1-13
- path shapes, cubic synonym 4-17 to 4-18, 4-38, 4-48
- patterns, synonym 4-17, 4-47 to 4-48
- 'pdoc' Apple event 2-40
- pen tables 4-20 to 4-21, 4-43 to 4-44
- portable digital document. *See also* print files
  - defined 1-5
- position constraint table 4-37, 4-41
- PostScript synonyms 4-12 to 4-14, 4-45
- 'ppnl' resource type. *See* panel resource
- precise-bitmap information
  - as format collection item 3-13
  - defined 3-93
- Print dialog box 1-10 to 1-12, 2-37 to 2-39, 2-73, 3-120
- Print Documents ('pdoc') Apple event 2-40
- printer driver name
  - as printer object property 4-7
  - retrieving 4-59
- printer drivers 1-8 to 1-9
- printer driver type
  - as printer object property 4-7
  - codes 4-7
  - retrieving 4-60
- printer name
  - as printer object property 4-7
  - retrieving 4-59

- printer object properties 4-6 to 4-7
  - job object 4-7
  - printer driver name 4-7
  - printer driver type 4-7
  - printer name 4-7
  - printer type 4-7
  - view device list 4-7
- printer objects 1-20, 4-6 to 4-9
  - color specification 4-8, 4-27 to 4-29, 4-84 to 4-89
  - counting view devices 4-57
  - defined 2-47
  - determining resolution for 4-26 to 4-27
  - manipulating properties of 4-54 to 4-61
  - manipulating view devices 4-56, 4-92
  - properties of. *See* printer object properties
  - retrieving from a job object 4-55
  - retrieving properties of 4-22 to 4-23
  - retrieving the driver name 4-59
  - retrieving the job object from 4-55
  - retrieving the printer driver type 4-60
  - retrieving the printer name 4-59
  - retrieving the printer type 4-61
  - retrieving view devices 4-57
  - selecting view devices 4-58
  - view devices 4-8, 4-25 to 4-29
- printers. *See* formatting printers; output printers; desktop printers
- printer type
  - as printer object property 4-7
  - retrieving 4-61
- print file object properties 4-10
  - format list 4-10
  - job object 4-10
  - page count 4-10
  - shape list 4-10
- print file objects 1-20, 4-9 to 4-10. *See also* print files
  - defined 2-47
  - determining job object 4-30, 4-64
  - manipulating properties of 4-61 to 4-70
  - properties of. *See* print file object properties
  - using 4-29 to 4-31
- print files 1-8
  - closing 4-29 to 4-30, 4-63
  - counting pages in 4-31, 4-65
  - deleting pages from 4-69
  - determining print-jobs 4-64
  - inserting pages in 4-68
  - opening 4-29 to 4-30, 4-62
  - QuickDraw picture data in 4-18 to 4-19
  - reading pages from 4-30 to 4-31, 4-65
  - replacing pages in 4-66
  - saving 4-30, 4-70
- printing 1-25 to 1-31, 2-61 to 2-68
  - and collection objects 3-7 to 3-14
  - and error handling 2-14 to 2-17
  - and idling 4-90
  - core objects 2-3 to 2-9
  - dialog box customization 3-22 to 3-27
  - direct-mode implementation 4-35 to 4-36
  - each page 2-64
  - finish a print job 2-65
  - finishing a page 2-67
  - forms 3-20
  - Gestalt selector 2-11, 2-47
  - halftone specifications 3-21 to 3-22, 4-15 to 4-16, 4-46
  - handling events 2-76
  - initializing the environment 2-11 to 2-12, 2-50
  - introduction 1-3 to 1-31
  - job format mode 4-19 to 4-20, 4-35 to 4-36, 4-39
  - object summary 1-20 to 1-22
  - of QuickDraw picture data 4-18 to 4-19
  - page at a time 2-21 to 2-22
  - page formatting 3-15 to 3-22, 3-39 to 3-66
  - print loop 2-20 to 2-24
  - resources 1-13, 3-24, 3-70 to 3-73, 3-127 to 3-132, 4-93 to 4-94
  - setting up the environment 2-50 to 2-51
  - shape-by-shape 2-22 to 2-24, 2-66
  - starting a print job 2-63
  - support for Finder 2-39 to 2-42
  - supporting dialog boxes 2-17 to 2-19
  - terminating the environment 2-11 to 2-12, 2-51
- printing errors
  - determining last 2-52
  - handling 2-14 to 2-17, 2-52 to 2-53
  - setting 2-53
- printing extensions 1-9 to 1-10
- Printing Manager, Macintosh
  - compatibility with QuickDraw GX 1-30, 2-75
  - printing documents created with 2-44 to 2-45
- printing modes 4-19 to 4-20. *See also* job format modes
- printing panel kinds 3-102
- printing phases 1-3 to 1-5
  - application 1-4
  - device communications 1-5
  - imaging 1-5
  - spooling phase 1-5
- printing-related collections 1-18 to 1-19. *See also* job collections; format collections; paper-type collections
  - accessing 3-28 to 3-33, 3-117 to 3-118
  - changing format items 3-112
  - collection tag ID 3-8, 3-77
  - identifying items 3-7 to 3-8
  - item categories 3-9, 3-76
  - items in 3-7 to 3-14
  - item structures 3-8
  - replacing items 3-31 to 3-33
- Printing Status dialog box 4-91

print-job information  
     as job collection item 3-10  
     defined 3-78  
 Print One Copy menu item 3-29  
 print-panel information  
     as job collection item 3-12  
     defined 3-88  
 print record conversion 2-45, 2-75

## Q

---

quality information  
     as job collection item 3-11  
     defined 3-83  
 QuickDraw GX Translator 2-45  
 QuickDraw picture data in print files 4-18 to 4-19

## R

---

reading pages from print files 4-65  
 reference constants  
     as job object property 2-6  
     retrieving from job objects 4-23 to 4-24, 4-52  
     setting in job objects 4-23 to 4-24, 4-53  
 resources  
     for dialog boxes 1-13, 3-24, 3-70 to 3-73, 3-127 to 3-132  
     retrieving from message handler 3-116  
 resource types  
     'DITL' 3-71  
     gxExtendedDITLType 3-72 to 3-73, 3-128 to 3-132  
     gxPrintPanelType 3-24, 3-70, 3-127  
     gxStatusType 4-93 to 4-94

## S

---

scaling information  
     as format collection item 3-13  
     custom panel for 3-116  
     defined 3-91  
 shape list, as print file object property 4-10  
 shape objects, and printing 1-23  
 special mapping information  
     as job collection item 3-11, 3-12  
     defined 3-87  
 spooling phase of printing 1-5  
 standard mapping information  
     as job collection item 3-11  
     defined 3-86

'stat' resource type. *See* status resource  
 status message 4-90 to 4-91  
 status resource 4-93 to 4-94  
 status structure 4-42  
 style name table 4-37, 4-41  
 synonyms 4-11 to 4-19. *See also* tag types  
     cubic 4-17 to 4-18, 4-38, 4-48  
     dash 4-14, 4-46  
     defined 1-5, 4-11 to 4-12, 4-45 to 4-49  
     halftone 4-15 to 4-16, 4-46  
     line cap 4-14 to 4-15, 4-47  
     pattern 4-17, 4-47 to 4-48  
     PostScript 4-12 to 4-14, 4-45  
     QuickDraw picture 4-18 to 4-19, 4-49  
     using 4-38

## T

---

tag objects  
     and printing 1-24  
     and synonyms 4-11 to 4-19  
 tag types  
     gxCubicSynonymTag type 4-18, 4-48  
     gxDashSynonymTag type 4-14, 4-46  
     gxFormatHalftoneTag type 4-15, 4-46  
     gxLineCapSynonymTag type 4-14, 4-47  
     gxPatternSynonymTag type 4-17, 4-47  
     gxPenTableTag type 4-21, 4-43  
     gxPostControlTag type 4-14, 4-45  
     gxPostScriptTag type 4-45  
     gxQuickDrawPictTag type 4-18, 4-49  
 text job format mode 4-36 to 4-37, 4-40 to 4-41. *See also*  
     job format modes  
 total overrides of printing messages 1-13  
 translated-document information  
     as job collection item 3-12  
     defined 3-89  
 tray-mapping information  
     as job collection item 3-12  
     defined 3-88

## U

---

unflattening  
     job objects 2-30 to 2-32, 2-58 to 2-60, 2-78  
     print-job flattening function 2-32  
     print-job from handle 2-30 to 2-31  
 units information  
     as paper-type collection item 3-14  
     defined 3-96

## V, W

---

- vector device pen tables 4-20 to 4-21, 4-43 to 4-44
- vector pen table entry structure 4-21
- vector pen table structure 4-21
- vertical page-flip information
  - as format collection item 3-13
  - defined 3-93
- view device mapping, for printers 4-8
- view device objects
  - and printing 1-25
  - printer usage 4-8
- view devices
  - as printer object property 4-7
  - counting for a printer 4-57
  - determining printer resolution 4-26 to 4-27
  - manipulating for printer objects 4-56, 4-92
  - retrieving for a printer 4-57
  - selecting for a printer 4-58
- view port objects, and printing 1-24

## X, Y, Z

---

- 'xdt1' resource type. *See* extended item list resource



---

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages were created on an Apple LaserWriter Pro printer. Final page negatives were output directly from text files on an Optrotech SPrint 220 imagesetter. Line art was created using Adobe Illustrator<sup>™</sup> and Adobe Photoshop<sup>™</sup>. PostScript<sup>™</sup>, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino<sup>®</sup> and display type is Helvetica<sup>®</sup>. Bullets are ITC Zapf Dingbats<sup>®</sup>. Some elements, such as program listings, are set in Apple Courier.

WRITERS

Gary McCue and Laine Rapin

DEVELOPMENTAL EDITOR

George Truett

ILLUSTRATORS

Ruth Anderson, Mai-Ly Pham

PRODUCTION EDITOR

Pat Christenson, Alan Morgenegg

PROJECT MANAGER

Trish Eastman

LEAD WRITER

David Bice

LEAD EDITOR

Laurel Rezeau

ART DIRECTOR/COVER DESIGNER

Barbara Smyth

Special thanks to Nik Bhatt, Tom Dowdy, Dennis Farnden, Dave Hersey, Ken Hittleman, Dan Lipton, Harita Patel, Amy Rosenstock, Ingrid Voss, Ron Voss, Sam Weiss, Chris Yerga

Acknowledgments to Betty Gee, Lorraine Findlay, Gary Hillerson, Marq Laube, Josephine Manuele, Barbara Martinez, Diane Patterson, Rich Pettijohn